
Z80-Maschinenprogramme mit Sharp MZ-700 und MZ-800
Helmut Ostermann
DJNZ
Z80
HC
Mein Home-Computer

HC – Mein Home-Computer

Helmut Ostermann

Z80-Maschinenprogramme mit Sharp MZ-700 und MZ-800

Eine Einführung mit vielen Beispielen



Informationen über den Bezug von Bändern mit den sieben umfangreichsten Beispielprogrammen (MENEBSDIS und sechs weitere) können bei der Firma Home-Computer-Systeme, Alexanderstr. 107, 2900 Oldenburg, angefordert werden.

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Ostermann, Helmut:
Z80-Maschinenprogramme [Z-achtzig-Maschinenprogramme] mit Sharp MZ-700 und MZ-800: e. Einf. mit vielen Beispielen / Helmut Ostermann. –
1. Aufl. – Würzburg: Vogel, 1985.
(HC – Mein Home-Computer)
ISBN 3-8023-0830-1

ISBN 3-8023-0830-1
1. Auflage. 1985

Umschlaggestaltung: Bernd Schröder, Böhl
Herstellung: Alois Erdl KG, Trostberg

Inhaltsverzeichnis

Vorwort

Dieses Buch will in die Kunst des Programmierens in Maschinensprache einführen. Es wäre zu wünschen, daß der Leser bereits in einer anderen Sprache programmiert hat oder zumindest eine Vorstellung davon hat, was Programmieren eigentlich ist. Die Kenntnis von Programmablaufplänen oder Struktogrammen wäre nützlich, ist aber nicht unbedingt erforderlich.

Das Buch will Anfänger anhand zahlreicher Beispiele in die Maschinensprache des Z80 einführen. Es verzichtet dabei ganz bewußt auf eine formale Systematik, die abstrakt die einzelnen Prozessorfunktionen abhandelt und die notwendigen Beispiele schuldig bleibt. Ein derartiges Buch könnte jedoch mit Gewinn (muß aber nicht) als Nachschlagewerk nebenbei benutzt werden. Wichtiger ist hingegen die Verfügbarkeit eines Z80-Handbuchs (siehe Anhang B). Notfalls kann man auch die Seiten 165 bis 168 des MZ-700-Handbuchs zu Rate ziehen, das in jedem Fall verfügbar sein sollte.

Konkrete Beispiele erscheinen aus folgenden Gründen für eine Einführung besonders wichtig:

1. Das Erlernen von Neuem nach abstrakten Regeln ohne Bezug auf (frühere oder aktuelle) Erfahrungen ist ineffektiv; ein Anfänger sollte jeden Schritt sofort am Computer nachvollziehen.
2. Das Programmieren in Maschinensprache erstreckt sich auf einen Problemkreis, der sich charakteristisch von dem mit höheren Sprachen bearbeiteten unterscheidet.
3. In der Praxis des Maschineprogrammierens greift man häufig auf die fertigen Routinen des Betriebssystems zurück. In den abstrakten Handbüchern fehlen wegen des nicht gegebenen Bezugs auf einen konkreten Computer die Hinweise hierauf fast völlig. Auf Beispiele wird in diesem Buch großer Wert gelegt; es soll sich gerade dadurch von abstrakten Darstellungen unterscheiden. Wir beschränken uns dabei auf das, was

von der Tastatur her machbar ist; das ist reichlich genug. Hardwarefragen und Probleme der Systemprogrammierung bleiben weithin ausgeklammert.

Lernziele sind:

1. Kennenlernen der wichtigsten Grundbegriffe;
2. Kennenlernen der wichtigsten Z80-Befehle;
3. Zurechtfinden in den Handbüchern;
4. Kennenlernen gängiger Programmstrukturen;
5. Anregungen für eigene Arbeit;
6. Anregungen zum Gebrauch von Dienstprogrammen;
7. Einblick in einige Programmiertricks.

Die Sprache des Mikroprozessors Z80 (bzw. Z80A) ist überall dieselbe, ganz gleich, in welchem Computer der Z80 arbeitet. Die Bedienung der Computer ist jedoch in der Regel sehr unterschiedlich, und ihre Betriebssysteme sind es auch. Der Verfasser wählte als Beispielmodell den MZ-700 aus, weil er ein gängiger Mikrocomputer ist, sich ausgezeichnet zum Arbeiten in Maschinensprache eignet und hervorragend dokumentiert ist.

Während der Drucklegung erschien der MZ-800 auf dem Markt. Er ist zwar – im Bezug auf die Maschinensprache – nicht so exzellent dokumentiert wie die 700er Serie, bietet dafür aber andere Vorzüge. Bei entsprechender Einstellung der Betriebsart (DIL-Schalter auf der Rückseite) ist er (ohne Aussage bezüglich Drucker) voll kompatibel zum MZ-700, so daß das Dargestellte auch für diesen schönen Personalcomputer gilt.

Wer mit einem anderen Z80-Computer arbeitet, wird sicherlich viele auch für ihn brauchbare Beispiele und Anregungen finden. Er muß jedoch selbst geeignete Speicherbereiche – und in vielen Beispielen die entsprechenden Routinen des Betriebssystems – heraussuchen.

Gehen wir nun ans Tun, fangen wir mit dem Einfacheren an, festigen wir es durch Beispiele und bauen wir Schritt für Schritt auf, doch immer so, daß sich eine konkrete Anwendung daraus ergibt. Dabei werden vor allem Verständnis und Anregung angestrebt. Die Perfektionierung einiger Beispiele bleibt dem Leser überlassen.

Oldenburg Helmut Ostermann

1
Die Sprache der Mikroprozessoren

Mikroprozessoren tun sehr schnell und sehr zuverlässig, was der Mensch ihnen vorschreibt, wenn ihnen diese Vorschriften in einer Form mitgeteilt werden, die sie verstehen. Sie reagieren auf elektrische Signale, die ihnen der Mensch z. B. über die Tastatur übermittelt. Hält man sich dabei streng an bestimmte Regeln, die durch die Bauweise des Prozessors bestimmt sind, kann man ihn zu sinnvollem Tun veranlassen. Man nennt auch dieses für zwei Partner verständliche Medium «Sprache».

1.1 Binär- und Hexadezimalzahlen

Schalt- und Steuerzentrale eines Mikrocomputers ist der Mikroprozessor, auch Central Processing Unit (CPU) oder Microprocess Unit (MPU) genannt. Beim Sharp MZ-700 ist dies ein Z80 (bzw. Z80A) der Firma Zilog, ein hochintegrierter Baustein von 13 mm Breite und 52 mm Länge (der eigentliche Chip ist sehr viel kleiner). Er hat 40 Anschlußbeinchen (Pins):

2 für die Versorgungsspannung;
1 für den Systemtakt (etwa 4 MHz);
13 lassen sich als Steuerleitungen bezeichnen;
8 sind Anfang des Datenbusses und
16 Anfang des Adreßbusses.

Für uns sind weiterhin nur die beiden letztgenannten Gruppen von Bedeutung: Der Datenbus besteht aus 8 Leiterbahnen (sozusagen Drähten), die entweder Spannung führen oder nicht. Vorhandensein von Spannung (genauer: eines im allgemeinen äußerst kurzen Impulses) wird dargestellt durch «1», Fehlen von Spannung (bzw. Impuls) durch

"0". Das macht es wesentlich leichter, den momentanen Zustand des Datenbusses zu beschreiben: Die Ziffernfolge 00101110 besagt eindeutig, daß der 1., 2., 3. und 5. «Draht» einen Impuls führen und der 0., 4., 6. und 7. nicht. Man numeriert hierbei die acht Leitungen von rechts nach links und zählt von 0 bis 7.

Es hat sich als äußerst zweckmäßig erwiesen, derartige Folgen der Ziffern 0 und 1 als einheitliche Zahlen anzusehen. Unser Zehnersystem kennt die zehn Ziffern 0, 1 ... 9. Es liegt nahe, ein Zahlensystem mit nur zwei Ziffern als Zweiersystem (häufiger Binär- oder Dualsystem) zu bezeichnen. Bekanntlich haben die Stellen einer Dezimalzahl unterschiedliches Gewicht; wir kennen – von rechts nach links – Einer, Zehner, Hunderter usw. Ganz entsprechend bedeuten die Stellen der Binärzahlen Einer, Zweier, Vierer, Achter, Sechzehner usw. Die Binärzahl 00101110 setzt sich dementsprechend zusammen aus

1 Zweier,
1 Vierer,
1 Achter und
1 Zweiunddreißiger.

Addiert man das, erhält man den Dezimalwert 46.

Die kleinste Dateneinheit, die ein Computer kennt, ist eine solche 0-oder-1-Stelle; man nennt sie Bit (engl.: Bißchen, auch erklärt als Zusammenziehung von binary digit, Zweierziffer). Mit dieser neuen Maßeinheit können wir sagen, daß der Datenbus unseres Computers 8 bit breit ist.

Der Datenbus nimmt die als Bitmuster codierten zu verarbeitenden Daten z. B. von der Tastatur (bzw. vom Tastenfeldcodierer) auf und leitet sie zum Prozessor, zum Speicher, zur Ausgabe usw. Obgleich man beim Programmieren in Maschinensprache ums Rechnen nicht herumkommt, brauchen wir nichts Schlimmes zu befürchten; wir haben nämlich fast schon mehr gerechnet, als eigentlich erforderlich war. Man zerlegt üblicherweise achtstellige Binärzahlen in zwei Vierergruppen und deutet jede für sich. Unsere Beispielzahl zerlegt man dementsprechend: 0010 1110. Über 15 brauchen wir im Binärsystem nicht hinauszurechnen!

Eine vierstellige Binärzahl kann 16 verschiedene Werte darstellen:

binär 0000 = hexadezimal 0 = dezimal 0
binär 0001 = hexadezimal 1 = dezimal 1
binär 0010 = hexadezimal 2 = dezimal 2
binär 0011 = hexadezimal 3 = dezimal 3
binär 0100 = hexadezimal 4 = dezimal 4
binär 0101 = hexadezimal 5 = dezimal 5
binär 0110 = hexadezimal 6 = dezimal 6
binär 0111 = hexadezimal 7 = dezimal 7
binär 1000 = hexadezimal 8 = dezimal 8
binär 1001 = hexadezimal 9 = dezimal 9
binär 1010 = hexadezimal A = dezimal 10
binär 1011 = hexadezimal B = dezimal 11
binär 1100 = hexadezimal C = dezimal 12
binär 1101 = hexadezimal D = dezimal 13
binär 1110 = hexadezimal E = dezimal 14
binär 1111 = hexadezimal F = dezimal 15

Woher und wozu nun die Buchstaben? Sie dienen hier lediglich als Ziffern, als Hexadezimalziffern (man sagt auch Sedezimalziffern, von lat. 16). Schon wieder ein neues Zahlsystem? Ja, aber wir werden mit ihm bald vertraut sein! Und ohne Kenntnis der Binär- und der Hexadezimalzahlen kann man einen Mikroprozessor, insbesondere den Z80, nicht programmieren! Schon äußerlich sind die Hexadezimalziffern eigentlich recht sympathisch: Zur Darstellung der aufgeführten 16 Werte kommen wir im Dezimalsystem nur zehnmal mit einer Ziffer aus, während wir in sechs Fällen zwei Ziffern benötigen. Das Hexadezimalsystem benötigt zur Darstellung dieser Werte stets nur eine Ziffer! Damit können wir unsere Beispielzahl auch so schreiben:

binär 0010 1110 = hexadezimal 2E = dezimal 46,
oder kürzer: 0010 1110b = 2Eh = 46d.

Zwei andere Beispiele:

0110 0100b = 64h = 100d,
1111 1111b = FFh = 255d.

Wir werden künftig die Kürzel b, h und d immer dann anbringen, wenn ihr Fehlen Verwechslungen verursachen könnte.

Überblick über die Stellenwerte (dezimal angegeben)

bin:	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
dez:	$10^3 = 1000$	$10^2 = 100$	$10^1 = 10$	$10^0 = 1$
hex:	$16^3 = 4096$	$16^2 = 256$	$16^1 = 16$	$16^0 = 1$

Keine Sorge! Höhere Werte benötigen wir nicht. Man kann aus der Aufstellung ersehen, daß man im Hexadezimalsystem den jeweils höheren Stellenwert durch Multiplikation mit 16 erhält. Wir können damit die Hexadezimalzahl 2E auch so deuten: 2 Sechzehner und 14 Einer, zusammen 46d.

Sie täten gut daran, sich die sechs neuen Ziffern recht bald einzuprägen. Mit einer mnemotechnischen Hilfe geht das vielleicht schneller:

A: 10 muß man sich so merken;
B: 11 man denke an das Kunstwort «*B*elf»;
C: 12 Rechtschreiben 6: «*C*wölf»;
D: 13 *D*reizehn;
E: 14 vi*E*rz*E*hn;
F: 15 *F*ünfzehn.

Wir können jetzt feststellen, daß der Adreßbus 16 bit breit ist. Um das zu verstehen, stellen wir uns den Speicher unseres Computers als ein Regal mit sehr vielen Fächern vor, die durchlaufend numeriert sind. Diese Fachnummern heißen Adressen. Das Bitmuster, das der Adreßbus bei einem Speicherzugriff gerade trägt, bestimmt, welche Speicherstelle angesprochen wird. Die niedrigste Adresse im Speicher ist 0000 0000 0000 0000b = 0000h, die höchste 1111 1111 1111 1111b = FFFFh. Wir könnten bequemer rechnen, halten uns aber eng an das Gesagte und rechnen so: FFFF = 15 * 4096 + 15 * 256 + 15 * 16 + 15 = 65535. Unser Mikroprozessor kann einen Speicher von 0000 bis FFFF «adressieren», also 65 536 verschiedene Speicherstellen ansprechen, deren jede Datenwerte von 00 bis FF (0 bis 255) aufnehmen kann. Die beiden Werte FF und FFFF sind typisch bzw. eine typische obere Grenze für einen Computer von der Größenordnung unseres Sharp.

Drei kleine Ergänzungen:

1. Man mache sich klar, daß man bei Verwendung von Dezimalzahlen nur 10 000 Speicherstellen hätte ansprechen können. Die Verwendung von Hexadezimalzahlen bedeutet also einen Gewinn von 555%!

2. Eine Gruppe von 8 bit heißt ein Byte. Der Inhalt eines Bytes läßt sich als zweistellige Hexadezimalzahl darstellen.
3. Wir setzen einmal die oben angefangene Zahlenreihe fort: 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024 ... Daß sich dabei eine Zahl in der Nähe von 1000 ergibt, veranlaßte die Informatiker, sie gleichfalls mit Kilo zu umschreiben, in Abkürzungen dieses Kilo jedoch mit einem großen K zu notieren. Man beachte den Unterschied:

1000 Meter = 1 Kilometer = 1 km;
1024 Byte = 1 Kilobyte = 1 KB.

Mit dieser Bezeichnungsweise berechnet man leicht: 65 536 Byte = 64 ⋆ 1024 Byte = 64 Kilobyte = 64 KB. Dieser Wert kennzeichnet also eine typische Größenklasse der Home- und Personalcomputer bzw. ihrer Speicher.

1.2 Was sind Maschinenprogramme, wozu dienen sie?

Wir sollten die Frage, welches denn wohl die verbreitetste Programmiersprache für Mikrocomputer sei, nicht vorschnell beantworten. Nicht jeder arbeitet in BASIC, aber alle arbeiten in einer Maschinensprache, auch wenn man sie in BASIC (oder Pascal oder Forth usw.) programmiert. Maschinensprache heißt die Sprache, in der der Mikroprozessor seine Befehle erhält. Auch wenn man den Computer in einer höheren problemorientierten Sprache programmiert, arbeitet seine CPU in ihrer Maschinensprache. Die Umwandlung aus einer höheren in eine Maschinensprache bewirken spezielle Hilfsprogramme, sogenannte Interpreter oder Compiler – auch das sind Maschinenprogramme!

Während man in einer höheren Programmiersprache z. B. mit Fließkommazahlen arbeiten, sie miteinander multiplizieren, die Quadratwurzel, den Sinus oder den Logarithmus berechnen kann, kann der Mikroprozessor nur ganz einfache Dinge, z. B. zwei ganze Zahlen addieren (oder subtrahieren), wobei das Ergebnis 255d = FFh nicht übersteigen darf; er kann Binärzahlen um eine Stelle nach links oder rechts verschieben (shiften) und viele derartige Elementaroperationen mehr. Und nun seine gewaltige Stärke: Er kann das unvorstellbar schnell, einige hunderttausend Elementaroperationen in einer Sekunde!

Wenn ein BASIC-Programm abläuft, zerlegt der Interpreter das Programm, ohne daß der Benutzer das merkt, sehr schnell in eine große Anzahl derartiger Elementarschritte, und der Mikroprozessor arbeitet das ihm so dargebotene Maschinenprogramm ab. Das geht zwar auch recht schnell, doch ohne den Umweg über den Interpreter ginge es einige hundert Male schneller. Diesen Unterschied können Sie drastisch erleben, wenn Sie das im Anhang A abgedruckte BASIC-Programm eintippen. Es enthält ein eingebundenes Maschinenprogramm.

Es kann nun – je nach Vorhaben – mehrere Gründe dafür geben, ein Problem in Maschinensprache zu programmieren: Um Morsezeichen zu dekodieren, wäre BASIC zu langsam. Man braucht dazu auch keine höhere Mathematik. Die Maschinensprache ist voll angemessen. Für die Darstellung schneller Bewegungen, z. B. eines Tänzers (vgl. Abschnitt 19.4) gelten die gleichen Überlegungen.

Um eine kleine Heizungsanlage zu steuern, braucht man keine neunstellige Genauigkeit. Die erforderlichen Wenn-dann-Beziehungen sind verhältnismäßig einfach. Man braucht dazu keine höhere Programmiersprache und keinen komplizierten Computer. Ein Einplatinenrechner tut's auch, und der arbeitet fast immer in Maschinensprache.

Dem, der Freude an der Technik und Sinn für Zahlen hat, macht der Einblick in die Arbeitsweise eines Mikrocomputers ganz einfach Spaß. Es gibt viele Aufgaben, die man in Maschinensprache lösen kann. Dazu will dieses Buch Einführung und Anregung bieten. Für den, der die Maschinensprache beherrscht, bietet sich ein weites Feld an Anwendungsmöglichkeiten: schnell bewegte Spiele, hochkomplizierte Spiele (z. B. Schach) oder auch eigene Programmiersprachen oder Varianten bestehender Sprachen lassen sich in Maschinensprache konzipieren.

Wie sieht nun Maschinensprache eigentlich aus? Wir geben ohne nähere Erklärung ein Beispiel, in dem die beiden Hexadezimalzahlen 1B und A4 addiert werden sollen:

kI	kI	Im	Im	Im	Im	Im	kI	0011 1110	3E
kI	kI	kI	Im	Im	kI	Im	Im	0001 1011	1B
Im	Im	kI	kI	kI	Im	Im	kI	1100 0110	C6
Im	kI	Im	kI	kI	Im	kI	kI	1010 0100	A4

Das sind drei verschiedene Darstellungsweisen dieses Problems. Links wird aufgezeigt, welche Impulskombinationen (Im = Impuls, kI = kein Impuls) dem Prozessor in etwa einer zweihundertneunzigtausendstel

Sekunde zugeführt werden müssen; das ist keineswegs besonders übersichtlich! Ein wenig besser ist da schon die Wiedergabe durch Binärzahlen in der Mitte. Am besten überschaubar ist die Darstellung durch Hexadezimalzahlen rechts. In dieser Form werden wir dem Mikroprozessor künftig mitteilen, was er tun soll. Manchmal müssen wir dabei jedoch auf die Einzelbitdarstellung der Binärzahlen zurückgreifen.

1.3 Ein Blick in den Speicher

Ab sofort benötigen wir den BASIC-Interpreter nicht. Sollte er geladen sein, schalten wir den Computer am besten aus und einige Sekunden später wieder ein. Es meldet sich der Monitor 1Z-013A. Das ist der Urlademonitor, ein äußerst wichtiges Hilfsprogramm. Es ist in einem ROM (read only memory) untergebracht, einem Speicherbaustein, aus dem man nur herauslesen, in den man aber nicht hineinschreiben kann (außer bei der Herstellung).

Unter der Meldung des Monitors sehen wir am linken Rand ein Sternchen, das Bereitschaftszeichen des Monitors, und daneben den schon von BASIC her bekannten Cursor.

Auch das Monitorprogramm (nicht zu verwechseln mit dem Bildschirmgerät) ist ein Maschinenprogramm. Es bewirkt zweierlei: 1. Es versetzt das Computersystem beim Einschalten in den erforderlichen Anfangszustand, so daß der Anlauf nicht nach irgendeinem wirren Muster erfolgt, das sich beim Einschalten gerade so oder so eingestellt hat. 2. Es enthält eine Fülle nützlicher Unterprogramme, die den Umgang mit dem Computer sehr vereinfachen. Wir werden einen Teil dieser Routinen für unsere Zwecke benutzen.

Das Monitorprogramm ist 4 KB lang und belegt die Speicherplätze 0000 bis 0FFF. Wir können es ohne Schwierigkeit sichtbar machen. Wir benutzen dazu den im Handbuch nicht aufgeführten ROM-Monitor-D-Befehl. Tippen Sie ein D0000⟨CR⟩, und Sie sehen die Inhalte der Speicherstellen

0000	0001	0002	0003	0004	0005	0006	0007
0008	0009	000A	000B	000C	000D	000E	000F
0010	0011	.....	.....	.....	.....	.....	.....
.....	.....	.....	.....	.....	.....	.....	.....
0098	0099	009A	009B	009C	009D	009E	009F

Wenn Sie nun den Befehl D0FC0 geben, sehen Sie den letzten Teil des Monitorprogramms und danach je vier Zeilen FF, 00, FF usw. In 1038 ... 103A steht jedoch C3 8D 03 – ein wichtiger Befehl – eingeschrieben. Dies ist der Arbeitsbereich des Monitors, ein Speicherbereich, in dem veränderliche Werte stehen, mit denen der Monitor arbeitet.

Nach D1200 sehen wir (von vereinzelten Ausnahmen abgesehen) abwechselnd je vier Zeilen FF und vier Zeilen 00. Diesen Bereich werden wir für die Unterbringung unserer Daten benutzen.

1.4 Veränderung des Speicherinhalts

Zur Veränderung der Speicherinhalte benutzen wir den auf Seite 149 des Handbuchs beschriebenen M-Befehl. Versuchen Sie, damit nach 1200 34 und nach 1201 56 einzugeben. Ganz gleich, ob Sie nun D11A0 oder D11B0 anfordern, in den beiden angesprochenen Speicherstellen stehen die eingegebenen Werte.

2
Erste Benutzung des Akkumulators

Wir schreiben nun unsere ersten einfachen Maschinenprogramme. Dazu müssen wir wissen, daß unser Mikroprozessor in sich (also neben dem bisher besprochenen Speicher) rund 20 Speicherstellen besitzt, die Register genannt werden und deren meiste 8 bit bereit sind. Das wichtigste dieser Register ist der Akkumulator (abgekürzt A), das Hauptrechenregister.

2.1 Laden mit einer Zahl, Laden von und nach Speicherstelle

Wir merken uns zunächst folgende Befehle:

LD A,n Lade den Akku mit einer Zahl n. (Weniger fachgerecht: Speichere die Zahl n in den Akku ein.)

LD A,(nn) Lade die in der Speicherstelle nn stehende Zahl in den Akku.

LD (nn),A Lade die im Akku stehende Zahl in die Speicherstelle mit der Adresse nn.

n ist dabei der Inhalt eines Bytes, eine zweistellige Hexadezimalzahl. Beispiele:

```
LD A,A7
LD A,(1200)
LD (1202),A
```

Zur Bedeutung der drei Bestandteile dieser Befehle merken wir uns das Schema WAS – WOHIN – WOHER:

WAS soll geschehen?
WOHIN mit der zu verarbeitenden Zahl (Operand)?
WOHER soll der Operand genommen werden (bzw. wer soll als Operand genommen werden)?

Unser erstes Programm soll die Zahl BF in den Akku laden und nach 1200 abspeichern; dann Akku mit 8E laden und nach 1201 abspeichern. Abschließend soll der in 1200 stehende Wert (nämlich BF) in den Akku geholt werden und nach 1202 und 1203 abgespeichert werden.

Die Aufgabe ist ganz einfach zu lösen. Wir notieren der Reihe nach

```
LD A,BF
LD (1200),A
LD A,8E
LD (1201),A
LD A,(1200)
LD (1202),A
LD (1203),A
HALT
```

Der letzte Befehl, *HALT*, ist nötig, damit der Prozessor nach Ablauf unseres Programms nicht unkontrolliert nach Maßgabe zufällig im Speicher stehender Zahlen weiterarbeitet, sondern sogleich seine Arbeit beendet. Wir werden schon bald einen bequemeren Programmabschluß kennenlernen.

Damit der Prozessor dieses Programm abarbeiten kann, muß er es in einer für ihn «verständlichen» Form verfügbar haben; man muß es ihm «eingeben». Daraus ergeben sich sofort zwei Fragen:

1. In welcher Form kann oder muß man das Programm eingeben?
2. In welchen Speicherbereich soll es eingegeben werden?

Zu 1: Wir schrieben unsere Befehlsliste in Form von Mnemonics, das sind leicht merkbare Kürzel (griech. mneme: Gedächtnis). Diese werden von einem Computer nicht ohne weiteres verstanden und müssen erst in eine ihm gemäße Form übersetzt werden. Dieser Vorgang heißt Assemblieren. Es gibt hochkomfortable Programme (Assembler), die das können. Sie sind nicht ganz billig und für einen Anfänger gar nicht leicht zu bedienen. Wir gehen ganz zünftig vor und erlernen zunächst das Assemblieren von Hand. Dazu benötigen wir Tabellen, die uns zu jedem Mnemonic den zugehörigen Opcode, eine Hexadezimalzahl, nen-

nen. (Siehe Anhang B, Literaturverzeichnis.) Die Opcodes der oben benutzten Befehle sind:

LD	A,n	3E
LD	A,(nn)	3A
LD	(nn),A	32
HALT		76

Es fehlen nun noch die Angaben über Operanden und Adressen. Einfache Operanden werden einfach angefügt. Aufpassen müssen wir bei den vierstelligen Adressen. Zunächst merken wir uns, daß man das höherwertige Byte (mit dem Stellenwert 256) mit *MSB* (most significant byte) und das niederwertige mit *LSB* (lower significant byte) bezeichnet. Beispiel: In 5A6F ist MSB=5A, LSB=6F. John von Neumann fand heraus, daß eine CPU etwas schneller arbeiten kann, wenn man ihr zunächst das LSB und dann erst das MSB darbietet. LD A,(5A6F) wäre demnach zu codieren 3A 6F 5A. Diese Vertauschung muß man sich ein für allemal einprägen; sie wird nicht nur beim Z80 angewendet.

Zu 2: Der Z80(A) wird dieses Programm in rund einer fünfzigtausendstel Sekunde abarbeiten. Kein Mensch wäre in der Lage, ihm diese Befehle in angemessener Zeit in einem Direktmodus zuzuführen. Deshalb geben wir die Befehle sozusagen auf Vorrat in den Speicher ein, von wo sie sich die CPU ohne Schwierigkeit in kürzester Zeit holen wird.

Für Maschinenprogramme im Clean Computer steht uns ein sehr großer Speicherbereich zur Verfügung, von 1200 bis CFFF. Das sind fast 48 KB, genauer: 48 640 Bytes. Wir wollen mit einprägsamen Adressen arbeiten und beginnen daher unser erstes Maschinenprogramm bei Adresse 2000. (Als Speicherbereich für die zu verarbeitenden Daten hatten wir 1200...1203 gewählt). Damit können wir endgültig codieren:

```
P2.1.1
2000  LD    A,BF        3E BF
2002  LD    (1200),A    32 00 12
2005  LD    A,8E        3E 8E
2007  LD    (1201),A    32 01 12
200A  LD    A,(1200)    3A 00 12
200D  LD    (1202),A    32 02 12
2010  LD    (1203),A    32 03 12
2013  HALT              76
```

Machen Sie sich unbedingt klar:

1. Die links angegebenen Speicherplätze errechnen sich aus der Startadresse, die fortlaufend um die Anzahl der Bytes eines jeden Befehls erhöht wird.
2. Das Programm ist 20 Bytes lang. Bitte zählen Sie das nicht nach, sondern errechnen Sie es aus Start- und Endadresse.
3. Bei 2-Byte-Operanden wird die Reihenfolge von MSB und LSB vertauscht.
4. Zu gleichen Mnemonics gehören gleiche Opcodes.

In unserer Aufstellung steht das eigentliche Maschinenprogramm in den (bis zu) drei Kolonnen rechts. Es besteht aus Zahlen, die den erforderlichen Speicherzustand beschreiben. Wir tippen sie in schon bekannter Weise ein: M2000, 3E, BF, 32, 00, ... usw. Zum Schluß drücken wir SHIFT-BREAK. Als letztes müssen wir das Programm starten, d. h. den Mikroprozessor veranlassen, sich der Reihe nach die Befehle aus dem Programmspeicher zu holen und abzuarbeiten. Das bewirkt der JUMP-Befehl J2000 (siehe Handbuch). In etwa einer fünfzigtausendstel Sekunde ist alles geschehen. Wir merken davon nichts und stellen nur verwundert fest, daß der Computer «hängt» und auf (fast) keine Taste mehr reagiert. Keine Panik, das ist richtig so, denn wir hatten ja das Programm mit HALT beendet, und der Prozessor hat seine Arbeit eingestellt. Aus dem groben Halt-Status kommen wir nur durch einen groben Eingriff wieder heraus: Auf der Rückseite unseres MZ-700 sitzt etwa 11 cm vom linken Gehäuserand entfernt ein kleiner viereckiger weißer Knopf, die Reset-Taste. Ihn drücken wir. Mit einem Piepton meldet sich der Monitor wieder, und das Prompt-Sternchen und der Cursor sind wieder da. Jetzt fordern wir mit D1200 einen Speicherauszug an und überzeugen uns, daß das Programm genau nach Plan abgelaufen ist. Die erste Zeile des Hexdumps muß jetzt beginnen: 1200 BF 8E BF BF. Vergleichen Sie das mit der Zielsetzung unseres ersten Programms.

2.2 Variante mit Einsprung in den Monitor

Wandeln Sie P2.1.1 in mehrfacher Weise ab; ändern Sie die zu verarbeitenden Zahlen (Operanden); verändern Sie in sinnvoller Weise die Adressen; fügen Sie auch noch einige Zeilen an! Und dann wollen wir die Blockade durch HALT vermeiden, wozu wir etwas weiter ausholen müssen: Wir hatten schon gehört, daß jeder Mikroprozessor zum Anlaufen ein Hilfsprogramm benötigt, das ihm vorschreibt, wie er das Computersystem initialisieren soll. Der Z80 ist so konstruiert, daß er den ersten Befehl dieses Anlaufprogramms in der Speicherstelle 0000 sucht. Wir veranlassen jetzt am Schluß unseres Programms die CPU, das Computersystem neu zu initialisieren, indem wir einen Sprung (jump) an die Speicherstelle 0000 befehlen. Statt HALT setzen wir nun

mnemonisch		codiert:
JP	*0000*	*C3 00 00*

Nach Abarbeiten des Programms springt der Prozessor an den Anfang des Urladeprogramms und läßt den Computer neu anlaufen. Der Speicherinhalt wird dabei nicht verändert.

Wenn alles wunschgemäß funktioniert, fordern Sie bitte noch einmal D0000 an. Sie können jetzt schon den Anfang des Monitorprogramms deuten:

0000	JP	004A	C3	4A	00
0003	JP	07E6	C3	E6	07
.....	...	.....	...	...	...

Etwa so beginnt der Monitor

Reset

0000	*Springe nach 004A*
004A	*Richte Stapel ab 10EF ein (vgl. Kapitel 10)*
004D	*Denke an mögliche Steuerimpulse von Drucker, Kassettenrekorder usw.*
004F	*Versetze die Bausteine, die z. B. die Tastatur abfragen, in die richtige Arbeitsstimmung.*
....	
....	
0078	*Lösche den Bildschirm.*
007D	*Mache den Hintergrund blau und die künftige Schrift weiß.*
....	
....	
009B	*Schreibe auf den Bildschirm * * MONITOR 1Z-013A * **
....	
....	

3
Weitere 8-bit-Register

Das wichtigste Register der CPU ist der Akkumulator. Daneben gibt es weitere, z. B. B, C, D, E, H und L. Sie können nicht alles, was der Akku kann, sind aber dennoch sehr wichtig und nützlich.

3.1 Die Befehle *LD r,r'* und ihre bitweise Codierung

Lassen Sie uns zunächst Daten zwischen den 8-bit-Registern hin- und hertransportieren. Dazu dient der LOAD-Befehl in der Form LD r,r', was bedeutet, daß der Inhalt des Registers r' in das Register r kopiert werden soll. Konkret sehen also die Mnemonics so aus: LD B,C oder LD A,H oder LD L,H usw. Der Opcode für diese Befehlsgruppe lautet 01......, worin jeder Punkt ein Bit bedeutet. Wir müssen nun bitweise programmieren. Zunächst die Codierung der genannten Register:

B: 000	*C: 001*	*D: 010*	*E: 011*
H: 100	*L: 101*		*A: 111*

Sie haben richtig bemerkt, daß man in 3 bit ja 8 Werte codieren kann, daß hier aber der Code 110 fehlt. Wir werden ihn erst später in einem anderen Zusammenhang kennenlernen.

Wenn wir beispielsweise LD B,A codieren wollen, müssen wir so vorgehen: 01 000 111, 01 für LD r,r', 000 für B und 111 für A. Das ordnen wir zu zwei Halbbytes (Nibbles): 0100 0111 und lesen es hexadezimal als 47.

A3.1.1 Codieren Sie ebenso a) LD D,A; b) LD B,C; c) LD C,B; d) LD H,L; e) LD L,H.

A3.1.2 Wenn man die Inhalte zweier Speicherstellen bzw. Register vertauschen will, benötigt man einen Zwischenspeicher. Vertauschen Sie die Inhalte von H und L, und benutzen Sie als Zwischenspeicher A! (Lösungen und Lösungsvorschläge finden Sie im Anhang C).

3.2 Addition, Subtraktion, Carry-Flag

Der Befehl *ADD A,r* veranlaßt die CPU, den Inhalt des Registers r zum Akku zu addieren. Dort steht dann auch das Resultat. Der Opcode hierfür ist 10000... .

A3.2.1 Codieren Sie a) ADD A,A; b) ADD A,B; c) ADD A,C; d) ADD A,D!

P3.2.2 Addition von D nach A

Die Inhalte der Speicherstellen 1200 und 1201 sollen addiert und die Summe nach 1202 abgelegt werden. Wir holen dazu (1200) – das ist der Inhalt von 1200 – in den Akku und transportieren ihn weiter in das Register D. Dann laden wir den Akku mit (1201) und addieren, um schließlich die Summe in 1202 abzulegen. Das Programm sieht so aus:

```
LD     A,(1200)
LD     D,A
LD     A,(1201)
ADD    A,D
LD     (1202),A
JP     0000
```

A3.2.2 Codieren Sie dieses Programm mit Beginn ab 2000.

Stimmt Ihre Lösung mit dem Hexdump L3.2.2 überein?

Wenn alles richtig ist, tippen Sie das fertige Programm in den Speicher ein, speichern mit M1200 zwei Summanden – z. B. 13 und 24 – ein und rufen das Programm mit J2000 auf. Der mit D1200 angeforderte Speicherauszug zeigt, daß der Computer richtig gerechnet hat: 1200 13 24 37.... Wir lassen weitere Beispiele rechnen und stutzen: 68 + 59 = C1. Wieso das? Lassen Sie uns nachrechnen: 8 + 9 = 17, hexadezimal ein Sechzehner und ein Einer; also muß das Ergebnis mit 1 enden. 6 + 5 = 11, mit Übertrag 12, geschrieben C. Ergebnis als C1. Rechnen Sie weitere Aufgaben, schrecken sie nicht vor Hexadezimalzahlen zurück, prüfen Sie die Ergebnisse nach, wie eben vorgeführt!

Noch ein Beispiel: 9A + AC ergibt 46; das stimmt doch wohl nicht! Rechnen wir nach: 10 + 12 = 22, also ein Sechzehner und sechs Einer; letzte Stelle 6 und dazu Übertrag merken. 9 + 10 + 1 = 20, also ein Sechzehner und vier Einer. Richtiges Ergebnis 146. Wo ist nur der Übertrag geblieben?

Hier muß verraten werden, daß es auch noch ein F-Register in der CPU gibt, ein Flag-Register. (Es hat nichts mit der fehlenden Codierung 110 von 3.1 zu tun). Eine Flagge ist ein Bit, das anzeigt, ob ein bestimmter Verarbeitungszustand eingetreten ist oder nicht. Das F-Register ist wie die bisher genannten 8 bit breit, könnte also bis zu 8 Flags repräsentieren. Es werden jedoch nicht alle 8 bit so genutzt. Eins davon ist das *Carry*-Bit (oder Carry-Flag, abgekürzt CY); es zeigt den Übertrag an. Der Übertrag in unserem letzten Beispiel ging nicht verloren, sondern wurde vom Prozessor sehr wohl für eine spätere Berücksichtigung «im Gedächtnis» behalten.

Man kann das Carry-Bit vom Programm aus manipulieren. Man setzt es mit dem Befehl *SCF* (set carry flag, Opcode 37), und man komplementiert es (d. h., man setzt es auf 1, wenn es auf 0 war – oder umgekehrt) mit dem Befehl *CCF* (3F).

Wir merken uns gleich noch einen nützlichen Befehl; er heißt *NOP* (no operation) mit dem Opcode 00. Er bewirkt nichts, hält aber Platz frei für einen Befehl, der vielleicht später einmal in das Programm eingefügt werden soll.

Addition mit Carry-Flag

Neben ADD gibt es Additionsbefehle, die in die Addition den Übertrag von einer vorhergehenden Addition einbeziehen. Wir lernen den ersten kennen:

mnemonisch:	codiert:
ADC A,r	10001 . . .

Wir gehen aus von P3.2.2, stellen uns jedoch vor, daß die CPU soeben eine Addition durchgeführt hat, aus der möglicherweise ein Übertrag zu berücksichtigen ist. Wir simulieren das, indem wir vor der Addition Carry setzen. Für eine spätere Variante lassen wir ein NOP als Platzhalter folgen. Außerdem üben wir das Codieren, indem wir statt des D-Registers das L-Register benutzen:

```
LD    A,(1200)
LD    L,A
LD    A,(1201)
SCF
NOP
```

```
ADC   A,L
LD    (1202),A
JP    0000
```

A3.2.3 Codieren Sie diesen Programmentwurf.

Lassen wir das fertige Programm zur Probe mit den Werten des vorigen Beispiels ablaufen, so erhalten wir «richtig falsch» 13 + 24 = 38; richtig deswegen, weil ja ein virtueller Übertrag von einer früheren Addition mitaddiert wurde. Um das zu vermeiden, müßten wir Carry löschen (auf null setzen). Wir werden später Möglichkeiten kennenlernen, wie das mit einem einzigen Befehl geschehen kann, nehmen zunächst einen zweiten Befehl in Kauf, um den Vorgang verständlicher zu machen. Wir setzen Carry wie bisher, ersetzen aber das folgende NOP durch CCF, komplementieren also das gesetzte Carry-Flag. Schon erhalten wir wieder 13 + 24 = 37. Bitte spielen Sie mit den aufgezeigten Möglichkeiten. Probieren Sie viele Beispiele aus, und schrecken Sie nicht vor echten Hexadezimalzahlen zurück. Sie können sich mit ihnen gar nicht besser vertraut machen als durch Ausprobieren vieler Möglichkeiten, die Sie sich selber ausdenken wollen.

3.3 Mehrstellige Addition und Subtraktion

Sechsstellige Hexadezimaladdition

Wir wollen nun zwei 3-Byte-Zahlen addieren. Das Ergebnis soll möglichst übersichtlich im Hexdump erscheinen. Deshalb benutzen wir die Speicherstellen

```
1200 1201 1202
1208 1209 120A
1210 1211 1212
```

Wenn wir also 123456 + 234567 = 3579BD berechnen wollen, soll der Speicherauszug so aussehen:

```
1200 12 34 56 .. .. .. .. ..
1208 23 45 67 .. .. .. .. ..
1210 35 79 BD .. .. .. .. ..
```

Wir addieren dazu zuerst die beiden LSBs mit ADD, dann die beiden mittleren Bytes und zum Schluß die beiden MSBs mit ADC.
A3.3.1 Versuchen Sie, das entsprechende Programm aufzustellen!

Wenn Sie die Aufgabe richtig gelöst haben, dann spielen Sie wieder. Benutzen Sie zur Addition auch einmal ein anderes Register, z. B. E oder H. Das Programm erscheint Ihnen für ein so einfaches Problem zu lang? Warten Sie's ab! Wir sind ja noch ganz am Anfang. Wir werden später mit kürzeren Programmen sechzehnstellig addieren.

Sechsstellige Dezimaladdition

Wir haben ganz bewußt mit der Hexadezimaladdition begonnen, denn unser Prozessor (und andere auch) hat nun einmal ein hexadezimales Innenleben. Er kann auch eine dezimale Summe bilden, doch ist das für ihn ein Sonderfall. Eine Voraussetzung für die korrekte Durchführung der Dezimaladdition ist, daß dem Prozessor nur die Dezimalziffern 0 bis 9 dargeboten werden. Nun rechnet die CPU (LSBs des vorigen Beispiels) zwar wieder 56 + 67 = BD. Läßt man aber den Befehl *DAA* (decimal adjust accumulator, Opcode 27) folgen, so wird der Inhalt des Akkumulators Stelle für Stelle in eine Dezimalzahl umgewandelt. Aus D wird 13, also eine 3 mit Übertrag zum linken Nibble, so daß sich dort 5 + 6 + 1 = 12 ergibt, also eine 2 mit Übertrag (im Carrybit). Dazu muß noch eine zweite Voraussetzung erfüllt sein: Es muß tatsächlich eine arithmetische Operation durchgeführt worden sein, damit – vom Benutzer unbeeinflußt – im F-Register das H-Bit (Halbbytecarry bei Übertrag vom 3. zum 4. Bit) richtig gesetzt wurde. Das fertige Programm mit Addition von H nach A sieht so aus:

```
P3.3.2
2000  LD   A,(1202)    3A 02 12
2003  LD   H,A         67
2004  LD   A,(120A)    3A 0A 12
2007  ADD  A,H         84
2008  DAA              27
2009  LD   (1212),A    32 12 12
200C  LD   A,(1201)    3A 01 12
200F  LD   H,A         67
2010  LD   A,(1209)    3A 09 12
2013  ADC  A,H         8C
2014  DAA              27
```

```
2015  LD    (1211),A    32 11 12
2018  LD    A,(1200)    3A 00 12
201B  LD    H,A         67
201C  LD    A,(1208)    3A 08 12
201F  ADC   A,H         8C
2020  DAA               27
2021  LD    (1210),A    32 10 12
2024  JP    0000        C3 00 00
```

Probieren Sie auch dieses Programm reichlich aus, aber zerstören Sie es nicht. Wir verwenden es sogleich als Gerüst eines neuen:

Sechsstellige Subtraktion

Die beiden hierzu neu benötigten Befehle sind

SBC A,r mit der Codierung 10011 ... und
SUB r mit der Codierung 10010 ...
(Bei R. Zaks [2] statt dessen SUB A,r)

Codieren Sie SUB A,H und SBC A,H und ersetzen Sie (in P6) in 2007 84 durch 94 und in 2013 und 201F jeweils 8C durch 9C.

A3.3.3 Wie gefallen Ihnen die Rechenergebnisse? Was haben wir unschön gemacht? Machen Sie es besser! Vergessen Sie nicht, auch hexadezimal zu subtrahieren; ersetzen Sie dazu die DAAs durch NOPs!

Rechnen Sie dann einmal 000000 – 000001. Nach allem, was wir bisher gehört haben, bedeutet das Ergebnis FFFFFFh = 16777215d, während offenbar −1 richtig ist. Hier lebt man mit einer Zweideutigkeit, die zu interpretieren Sache des Programmierers ist: Wenn das höchst Bit einer Zahl gesetzt ist (im Beispiel Bit 23), faßt man sie häufig als negativ auf. Wir kommen darauf zurück, merken uns aber schon, daß man von einer positiven Zahl zur entsprechenden negativen gelangt, indem man ihre Bits komplementiert (aus 0 mach 1, aus 1 mach 0) und dann 1 addiert. Zu unserem Beispiel:

+1 = 0000 0000 0000 0000 0000 0001
−1 = 1111 1111 1111 1111 1111 1110 + 1
= F F F F F F

4
Einige 16-bit-Register, insbesondere HL

Die Register B, C, D, E, H und L lassen sich mit einfachen Befehlen als 16-bit-Doppelregister ansprechen: BC, DE, HL. Die Bezeichnungsweise entspricht der normalen Schreibweise der Zahlen: Das MSB befindet sich in B, D oder H und das LSB in C, E oder L.

4.1 Die Befehle *LD dd,nn*

Die Befehle dieser Gruppen laden das jeweils codierte Doppelregister dd mit der vierstelligen Zweibytezahl nn. Ihre Codierung ist 00dd0001, worin für dd folgende Codes einzusetzen sind:

BC: 00 *DE: 01* *HL: 10*

Wir überzeugen uns von der Richtigkeit des Gesagten, indem wir z. B. das HL-Register mit ABCD laden und dann den Inhalt von H über A nach 1200 und den Inhalt von L über A nach 1201 transferieren. Wir lassen sofort ein zweites Testprogramm folgen, das dasselbe mit der Zahl 6789 bewirkt:

A4.1.1 Wo steht was im HL-Register?

a) ab 2000		b) ab 200E	
LD	HL,ABCD	LD	HL,6789
LD	A,H	LD	A,H
LD	(1200),A	LD	(1200),A
LD	A,L	LD	A,L
LD	(1201),A	LD	(1201),A
JP	0000	JP	0000

Codieren Sie und vergleichen Sie mit L4.1.1; rufen Sie dann den ersten Teil mit J2000 auf und kontrollieren Sie den Hexdump. Machen Sie

dasselbe mit J200E und überzeugen Sie sich, daß alles nach Wunsch verläuft. Vielleicht fügen Sie noch einen dritten Abschnitt an?

c) Vereinfachung?

4.2 Die Befehle *LD (nn),dd* und *LD dd,(nn)*

Die Befehle dieser Gruppen kopieren den Inhalt eines Doppelregisters in zwei benachbarte Speicherstellen bzw. umgekehrt. Die zugehörigen Opcodes sind ED 01dd0011 bzw. ED 01dd1011.

Der folgende Dreizeiler holt die Inhalte von 1200 und 1201 in das BC-Register und legt sie dann in 1202 und 1203 ab. (Eine Tafel Schokolade, die man aus dem Schrank holt, ist nicht mehr im Schrank. Holt man eine Zahl aus dem Speicher oder einem Register, so bleibt sie im Ursprung erhalten: Die Zahl wird kopiert.)

```
P4.2.1
2000  LD   BC,(1200)      ED 4B 00 12
2004  LD   (1202),BC      ED 43 02 12
2008  JP   0000           C3 00 00
```

Sie haben richtig beobachtet: Die Maschinenbefehle des Z80 können bis zu vier Bytes lang sein.

Das HL-Register bietet gegenüber den anderen 16-bit-Registern mancherlei Vorzüge. So gibt es zu den beiden neuen LD-Befehlen zwei spezielle Varianten für das HL-Register:

LD (nn),HL	(Opcode 22)
LD HL,(nn)	(Opcode 2A)

A4.2.2 Führen Sie a) mittels DE-Register, b) mittels HL-Register folgenden Speicherzustand herbei:

1200   01 23 45 67

4.3 Vierstellige Hexadezimaladdition

Sie werden sicherlich fragen, wozu denn nun vierstellig, wo wir doch schon sechsstellig addiert haben. Antwort: Hier sollen alle vier Stellen auf einmal addiert werden. Dazu zwei neue Befehle zur Auswahl:

ADD HL,ss	00ss1001
ADC HL,ss	ED 01ss1010

Für ss gilt die gleiche Codierung wie für dd (vgl. Abschnitt 4.1). Die Summanden sollen in von Neumannscher Anordnung in 1201 und 1200 und in 1209 und 1208 bereitgestellt sein. Ablage des Ergebnisses nach 1211 und 1210:

```
P4.3.1
2000  LD   HL,(1200)    2A 00 12
2003  LD   DE,(1208)    ED 5B 08 12
2007  ADD  HL,DE        19
2008  LD   (1210),HL    22 10 12
200B  JP   0000         C3 00 00
```

A4.3.2 Schreiben Sie ein möglichst ähnliches Programm, bei dem die Zahlen an denselben Stellen, jedoch in gut lesbarer Anordnung (MSB links, LSB rechts) vorgegeben werden.

A4.3.3 Subtrahieren Sie P4.3.1 entsprechend zwei vierstellige Zahlen ohne Carry voneinander. Suchen Sie den benötigten Befehl im Handbuch!

A4.3.4 Verachtfachen Sie Zahlen < 2000h durch dreimaliges Verdoppeln.

4.4 Indirekte Adressierung

Laut Krimi geht man bei der Übergabe eines Lösegeldes meistens so vor, daß man den Überbringer zu einem angeblichen Treffpunkt bestellt, ihm dort aber die Nachricht zukommen läßt, daß er sich anderswohin begeben möge. Genauso ist es bei der indirekten Adressierung: Man «sagt» dem Prozessor, wo er die endgültige Adresse findet. Dazu benutzt man häufig das Doppelregister HL. Drei neue Befehle:

LD (HL),A kopiert den Inhalt des Akkus in die in HL angegebene Speicherstelle.

LD (HL),n lädt die Speicherstelle, deren Adresse in HL steht, mit der Konstante n.

INC HL erhöht den Inhalt von HL um 1.

Die Codierung entnehmen Sie bitte dem Handbuch. Übrigens, (HL): 110 ist die fehlende Codierung von 3.1.

A4.4.1 Der Speicherbereich von 1200 bis 1207 soll mit Hilfe der neuen Befehle (und mit viel Fleiß) auf null gesetzt werden. Versuchen Sie's!

5
Bedingte und unbedingte relative Sprünge

Man unterscheidet Sprünge, die nur dann ausgeführt werden, wenn eine bestimmte Bedingung erfüllt ist, und solche, die unabhängig von einer Bedingung immer ausgeführt werden. Sie heißen «bedingte» bzw. «unbedingte» Sprünge. Ist nicht das Sprungziel angegeben, sondern die Sprungweite, heißen sie «relative» Sprünge.

5.1 Der Superbefehl *DJNZ*

Sie werden in L4.4.1 mit Recht die häufige Wiederholung mancher Befehle als unschön empfunden haben. Wir wollen jetzt den Speicherbereich von 1200 bis 129F auf null setzen, jedoch nicht so primitiv. Wir benutzen dazu den überaus geschickt konstruierten Befehl DJNZ (decrement and jump if not zero). Er arbeitet mit dem B-Register zusammen.

Rechnen Sie nach: Es sind 160 Speicherstellen auf null zu setzen, also hexadezimal A0. Wir laden das B-Register mit A0, HL mit 1200 und lassen LD (HL),00, INC HL und den Befehl DJNZ folgen. Er vermindert den Inhalt des B-Registers um 1, und wenn es noch nicht den Wert null erreicht hat, veranlaßt er den Prozessor, das Ganze so lange zu wiederholen, bis B = 0. Damit läßt sich unser Ziel wie folgt erreichen:

```
P5.1.1
2000  LD    B,A0          06 A0
2002  LD    HL,1200       21 00 12
2005  LD    (HL),00       36 00
2007  INC   HL            23
2008  DJNZ  ....          10 ..
200A  JP    0000          C3 00 00
```

5.2 Die Codierung der Sprungweite

Wir haben nun zweierlei zu überlegen:
1. Wohin, zu welchem Befehl soll der Prozessor in P5.1.1 nach Anweisung 2008 springen, um den Programmablauf fortzusetzen, wenn B noch nicht gleich null geworden ist?
2. Wie ist dieser Sprung zu codieren?
Zu 1. (Mindestens) eine Speicherstelle wurde auf null gesetzt, und (HL) wurde um eins erhöht. Wenn noch nicht der ganze Speicherbereich auf null gesetzt wurde, so ist jetzt die nächste Speicherstelle auf null zu setzen. Es muß also nach 2005 zurückgesprungen werden. Man nennt eine solche Programmstruktur «Schleife». Sie wird hier ausgelöst durch den bedingten Sprung DJNZ, der nur dann erfolgt, wenn B ≠ 0. Man sagt auch, daß in 2008 eine «Verzweigung» erfolgt.
Zu 2. In Übereinstimmung mit zahlreichen Assemblern und Disassemblern geben wir beim Mnemonic die Zieladresse «absolut» an: DJNZ 2005. Im Maschinenprogramm (Objektcode) muß dem Prozessor jedoch die relative Sprungweite mitgeteilt werden. Dazu muß man wissen, daß während der Bearbeitung eines Befehls der Programmzähler, ein bisher noch nicht erwähntes Register der CPU, bereits auf den nächsten zeigt; bei der Bearbeitung des Befehls 2008 also schon auf 200A. Wir berechnen damit die Sprungweite so: 200A − 2005 = 05. Nun erfolgt dieser Sprung aber rückwärts; relative Sprungweite also −05. Leider können wir dem Prozessor das Minuszeichen nicht direkt eingeben, sondern müssen – wie zu P3.3.3 erklärt – so rechnen:

+5 = 0000 0101
−5 = 1111 1010 + 1 = 1111 1011 = FB.

Die Programmzeile 2008 lautet also vollständig

2008 DJNZ 2005 10 FB

Wir hätten damit die wohl größte Schwierigkeit, die beim Programmieren in Maschinensprache auftritt, bewältigt. Sie täten gut daran, die angezeigten Schritte sorgfältig nachzuvollziehen. Vielleicht trägt die folgende Tabelle noch ein wenig zum Verständnis bei: Von 200A aus (= Absprungadresse + 2 = Programmzählerstand) bewirkt

FA = −6 einen Sprung nach 2004
FB = −5 einen Sprung nach 2005
FC = −4 einen Sprung nach 2006
FD = −3 einen Sprung nach 2007
FE = −2 einen Sprung nach 2008
FF = −1 einen Sprung nach 2009
00 = 0 einen Sprung nach 200A
01 = +1 einen Sprung nach 200B
02 = +2 einen Sprung nach 200C usw.

Vielleicht gefällt Ihnen folgende Anleitung besser: Unterscheiden Sie zwischen Vorwärtssprung (in Richtung des allgemeinen Programmablaufs) und Rückwärtssprung (wie in unserem Beispiel).

Vorwärtssprung (im allgemeinen nicht mit DJNZ!):
Absprungadresse (Adresse des Sprungbefehls) = 23D3
PC (program counter, Programmzähler) = 23D5
Sprungziel (Sprungadresse) = 23EB
23EB − 23D5 = 16
Relative Sprungweite (Codierung) = 16.

Rückwärtssprung:
Absprungadresse (Adresse des Sprungbefehls) = 2008
PC = 200A
Sprungziel = 2005
(2005 + 100) − 200A = FB
Relative Sprungweite (Codierung) = FB.

Da die Sprungweiten in einem einzigen Byte codiert werden, ergibt sich folgende Begrenzung:

Bei Vorwärtssprüngen max. 7F = 127d;
Bei Rückwärtssprüngen max. 80 = −128d;
min. FF = −1d.

A5.2.1 Berechnen Sie die Codierung folgender Sprünge:

a) von 653E nach 65BF
b) von 653F nach 65BE
c) von 20A7 nach 2110
d) von 20CB nach 20BB

e) von 32F6 nach 3299
f) von BFAA nach BE2A
g) von BFAA nach BF23

Als Absprungadresse wurde dabei die Adresse des Sprungbefehls angegeben.

5.3 Anwendung: Das kleine Einmaleins

Wir wollen sogleich den Befehl DJNZ auf ein anderes Problem anwenden und Produkte aus dem Bereich des kleinen Einmaleins (bis F * F = E1) berechnen. Den einen Faktor holen wir aus 1200 ins B-Register, den anderen aus 1201 ins C-Register. Der Akku wird anfangs auf null gesetzt. Dann addieren wir den Inhalt von C so oft zu A, wie das B-Register vorschreibt. Das fertige Produkt legen wir in 1202 ab.

```
P5.3.1
2000  LD    A,(1200)     3A 00 12
2003  LD    B,A          47
2004  LD    A,(1201)     3A 01 12
2007  LD    C,A          4F
2008  LD    A,00         3E 00
200A  ADD   A,C          81
200B  DJNZ  ....         .. ..
200D  LD    (1202),A     32 02 12
2010  JP    0000         C3 00 00
```

A5.3.1 Fragen:
a) Wohin muß der Rücksprung erfolgen?
b) Was enthält der Programmzähler beim Absprung?
c) Wie ist die Sprungweite zu codieren?
d) Wie lautet Zeile 200B vollständig?

Das Programm arbeitet zuverlässig, obwohl es einen Denkfehler enthält. Erkennen Sie ihn? Berechnen Sie 0 * 9! Das Ergebnis stimmt, und doch haben wir falsch gerechnet! Was geschieht im B-Register? Der Fehler hebt sich selbst auf, so daß wir im Augenblick darüber hinweggehen wollen. Wir kommen aber schnell darauf zurück.

5.4 Weitere Sprungbefehle

Hier sind vor allem folgende Befehle zu nennen:

JR C,e	mit Opcode 38
JR NC,e	mit Opcode 30
JR Z,e	mit Opcode 28
JR NZ,e	mit Opcode 20

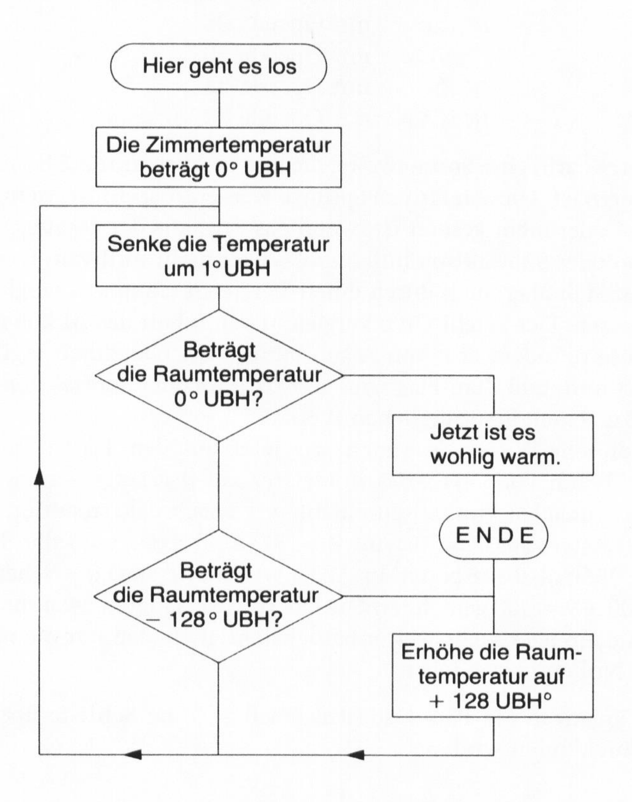
e bedeutet darin eine Sprungweite, die wie in Abschnitt 5.2 besprochen zu codieren ist. Diese relativen Sprünge werden ausgeführt, wenn Carry gesetzt – oder nicht gesetzt ist, wenn das Ergebnis der voraufgehenden Addition oder Subtraktion null (zero) – oder nicht null war. Das Zero-Flag läßt sich aber auch durch einen Vergleich (compare, vergleichen) beeinflussen. Der Befehl *CP n* vergleicht den Inhalt des Akkus mit der Konstante n, indem er n von A subtrahiert. Bei Gleichheit ergibt sich der Wert null, und Zero-Flag wird gesetzt. Trotz der Subtraktion behält der Akku seinen ursprünglichen Wert.

Mit diesem Wissen kommen wir jetzt auf den Fehler in P5.3.1 zurück: Wenn vor DJNZ das B-Register auf 0 gesetzt war, wird die Schleife (zunächst einmal) durchlaufen. Danach dekrementiert DJNZ das B-Register; aus B = 0 wird B = FF. Und dann wird die Schleife weitere 255mal abgearbeitet. In P5.3.1 wurde also statt 0 ⋆ 9 (hexadezimal!) 100 ⋆ 9 = 900 gerechnet. Die Überträge gelangten zwar ins Carry, wurden dann aber nicht weiter berücksichtigt, so daß nur die nachfolgenden Nullen übrigblieben.

A5.4.1 Ergänzen Sie P5.3.1 so, daß bei B = 0 die Schleife überhaupt nicht durchlaufen wird.



Eine bedenkliche Heizungsregelung



So ähnlich haben wir in P5.3.1 mit null multipliziert.
NB: °UBH ist die neue, nach dem Kalobiotiker Prof. Unbehagen benannte Temperaturskala: 0°UBH ist die Temperatur, bei der sich ein 1,72 m großer und 72 kg schwerer Nichtraucher am wohlsten fühlt.



6
Unterprogramme (Subroutinen)

Unterprogramme sind in sich abgeschlossene Programmteile, die im allgemeinen häufiger durchlaufen, aber nur einmal geschrieben werden. Sie werden im einfachsten Fall mit *CALL nn* aufgerufen und enden mit *RET* (return).

Bei CALL (Opcode CD) geschieht folgendes: Der Prozessor «merkt» sich die Adresse des an sich folgenden Befehls, vollführt aber einen Sprung zur angegebenen Adresse des Unterprogramms. Findet er in dessen Verlauf einen RET-Befehl, kehrt er zu der «gemerkten» Adresse zurück und arbeitet von dort ab weiter – so als wäre nichts geschehen.

6.1 Aufruf mit CALL nn

Um das Konzept der Unterprogramme an einem einfachen Beispiel anzuwenden, schreiben wir (wenngleich nicht gerade sinnvoll) P5.3.1 so um, daß formal eine Subroutine (2015 ... 2016) entsteht:

```
P6.1.1
2000  LD    A,(1200)       3A 00 12
2003  LD    B,A            47
2004  LD    A,(1201)       3A 01 12
2007  LD    C,A            4F
2008  LD    A,00           3E 00
200A  CALL  2015           CD 15 20
200D  DJNZ  200A           10 FB
200F  LD    (1202),A       32 02 12
2012  JP    0000           C3 00 00
2015  ADD   A,C            81
2016  RET                  C9
```

Die Subroutine 2015 wird nun so oft durchlaufen, wie der Inhalt der Speicherstelle 1200 angibt.

6.2 Betriebssystem, Monitor

Ein Betriebssystem ist ein Programm, das die einzelnen Komponenten eines Computers überwacht und koordiniert. Es besteht aus vielen Unterprogrammen, die Einzelaufgaben wie z. B. die Tastaturüberwachung (Tastaturabfrage), Auffinden der nächsten Schreibstelle auf dem Bildschirm usw. erledigen. Es ist allgemein üblich, diese Subroutinen um des besseren Verständnisses willen mit Kurznamen zu bezeichnen. Vom Programm her werden sie jedoch mit ihrer Startadresse aufgerufen.

Der schon mehrfach angesprochene ROM-Monitor ist im wesentlichen ein solches Betriebssystem. Seine Routinen werden sehr oft von Anwenderprogrammen mitbenutzt. Wir lernen die ersten kennen:

Die Subroutine BELL (003E) löst ein kurzes Tonsignal aus. Wir können sie sehr schnell anwenden, indem wir in P6.1.1 Zeile 200A abändern zu CALL 003E. Dann piept der Computer so oft, wie der Inhalt von 1200 vorschreibt, und noch einmal hinterher beim Rücksprung in den Monitor.

6.2.1 Verschachtelung von Subroutinen

Wir können die Subroutine BELL aber auch an anderer Stelle unterbringen. Wir stellen in P6.1.1 die ursprüngliche Fassung des Befehls 200A (CALL 2015) wieder her und schreiben die Subroutine so:

```
P6.2.1
2015  ADD  A,C       81
2016  CALL 003E      CD 3E 00
2019  RET            C9
```

Jetzt wird die Subroutine BELL aus einer anderen Subroutine heraus aufgerufen. Man darf Subroutinen ineinander verschachteln; je nach Speicherausnutzung bis zu einer Tiefe von rund hundert Ebenen – und mehr (vgl. Abschnitt 10.4).

6.3 Erweiterung des Betriebssystems (EBS)

Manche der folgenden Programme werden sich für die Praxis des Programmierens in Maschinensprache als so nützlich erweisen, daß wir sie für spätere Benutzung unter der Kurzbezeichnung «EBS» bereitstellen wolllen. Hinter dem Namen EBS sollten wir keine exakte Klassifizierung vermuten; wir werden auch einige Dienstprogramme (Utilities) unter diesem Namen aufzeichnen.

6.3.1 Die EBS-Routine NULL12

Wenn wir Maschinenprogramme testen wollen, ist es oft wünschenswert, einen bestimmten Speicherbereich, z. B. den, in dem wir unsere Daten unterbringen wollen, auf einen übersichtlichen Anfangszustand zu bringen, z. B. den Bereich 1200...129F auf den Wert 00. Das hat bereits P5.1.1 getan. Wir übernehmen nun dieses Programm als Routine ins EBS. Zweierlei ist zu überlegen:

1. Wo soll das EBS untergebracht werden? Da wir unseren Arbeitsbereich und unsere Maschinenprogramme gewohnheitsmäßig weit nach vorn verlegen, liegt es nahe, den Speicher von hinten her mit dem EBS zu belegen. Wir beginnen jedoch nicht mit CFFF als Endadresse (vgl. Sharp-Handbuch Seite 126), sondern mit AFFF, um Platz zu lassen für einen Disassembler.
2. Ein EBS hat nur dann Sinn, wenn es immer verfügbar ist, wenn es benötigt wird. Wir werden es auf Band aufzeichnen und immer dann laden, wenn wir eigene Maschinenprogramme schreiben wollen.

Als erstes schreiben wir P5.1.1 so um, daß es mit RET auf Platz AFFF endet:

```
P6.3.1
NULL12  AFF4  LD    B,A0       06 A0
        AFF6  LD    A,00       3E 00
        AFF8  LD    HL,1200    21 00 12
        AFFB  LD    (HL),A     77
        AFFC  INC   HL         23
        AFFD  DJNZ  AFFB       10 FC
        AFFF  RET              C9
```

Wir können NULL12 auch manuell – ohne Programmkontrolle – benutzen indem wir JAFF4 eintippen. Der Computer kehrt dann allerdings auf u. U. abenteuerlichen Wegen in den Monitor zurück.

Subroutinen können recht flexibel sein. NULL12 erlaubt es z. B., den gewünschten neuen Inhalt der Speicherstellen in A, die Anzahl der zu überschreibenden Speicherstellen in B und die Adresse der zuerst zu überschreibenden Speicherstelle in HL wahlweise vorzugeben. Der Einsprung erfolgt dann bei AFFB.

A6.3.2 Der Speicherbereich von 1230 bis 126F soll auf FF gesetzt werden.

Wir zeichnen jetzt NULL12 auf Band auf. Dazu nehmen Sie am besten ein neues Band oder ein Band, das überschrieben werden darf und stets für diesen Zweck zur Verfügung stehen soll. Spulen Sie es zurück, so daß es von der gewünschten Kassettenoberseite aus gesehen links sitzt. Drehen Sie dann das Band mit einem dicken Bleistift oder Kugelschreiber so weit vor, daß der Vorspann (wenn vorhanden) gerade rechts zu verschwinden beginnt. Legen Sie das Band ins Kassettenlaufwerk. Wir stellen fest:

1. Das, was «gesaved« werden soll, beginnt z. Z. in AFF4
2. und endet mit AFFF.
3. Nach dem Wiederladen von Band soll der Computer in den Monitor springen, also nach 0000.

Dementsprechend geben wir jetzt den ROM-Monitor-Befehl SAFF4AFFF0000 und drücken ⟨CR⟩. Der Computer fragt FILENAME? Antworten Sie mit EBS. Alles andere können Sie im Handbuch nachlesen. Spulen Sie nach dem Saven nicht zurück, sondern saven Sie gleich anschließend noch einmal (Sicherheitsmaßnahme). Schalten Sie den Computer aus und wieder ein und laden Sie NULL12 vom Band. Mit JAFF4 können Sie sich überzeugen, daß alles korrekt abläuft.

7
Ausgabe von Zeichen und Strings

Ein Byte kann Werte von 00 bis FF beinhalten; man kann sie – je nach Zusammenhang – als Zahlen oder als codierte Zeichen deuten. Die Zuordnung erfolgt in aller Regel nach dem American Standard Code for Information Interchange (ASCII). Machen Sie sich unbedingt mit dem ASCII vertraut (Seite 158 des Sharp-Handbuchs).

Eine Folge von Zeichen heißt Zeichenkette oder String.

7.1 Die Monitorroutine PRNT

PRNT (ab 0012) schreibt das Zeichen auf den Bildschirm, das codiert im Akku steht. Lesen Sie auf Seite 152 des Sharp-Handbuchs nach!

Wir wenden sie sogleich in einem Trivialprogramm an:

```
P7.1.1
2000  LD    A,43        3E 43       ;'C'
2002  CALL  0012        CD 12 00
2005  LD    A,B7        3E B7       ;'o'
2007  CALL  0012        CD 12 00
200A  LD    A,B3        3E B3       ;'m'
200C  CALL  0012        CD 12 00
200F  LD    A,9E        3E 9E       ;'p'
2011  CALL  0012        CD 12 00
2014  LD    A,A5        3E A5       ;'u'
2016  CALL  0012        CD 12 00
2019  LD    A,96        3E 96       ;'t'
201B  CALL  0012        CD 12 00
201E  LD    A,92        3E 92       ;'e'
2020  CALL  0012        CD 12 00
2023  LD    A,9D        3E 9D       ;'r'
2025  CALL  0012        CD 12 00
2028  HALT              76
```

Wenn Sie mit dieser Fleißlösung nicht einverstanden sind, haben Sie schon einen recht guten Programmiergeschmack entwickelt. Wir fügen denn auch sogleich die schon bekannten Befehle zu einer eleganteren Lösung zusammen:

1. Der auszugebende String wird ab 1200 bereitgestellt.
2. Von dort holen wir die einzelnen Zeichen mittels LD A,(HL) in den Akku.
3. Vorher ist das B-Register mit der Anzahl der auszugebenden Zeichen zu laden.
4. Nach Ausgabe durch PRNT und INC HL Wiederholung bis zum Ende des Strings.

```
P7.1.2
1200                   43 B7 B3 9E A5 96 92 9D
3000  LD    HL,1200    21 00 12
3003  LD    B,08       06 08
3005  LD    A,(HL)     7E
3006  CALL  0012       CD 12 00
3009  INC   HL         23
300A  DJNZ  3005       10 F9
300C  HALT             76
```

A7.1.2 Wandeln Sie das Programm so ab, daß der String mit der Anzahl der nachfolgenden gültigen Zeichen beginnt. Sein erstes Byte muß natürlich nach B geholt werden.

Ihre Lösung bietet den Vorteil, daß man ab 1200 beliebige Strings bereitstellen und mit einheitlichem Programm ausgeben kann. Sie sollten das mehrfach ausprobieren!

Wozu aber zählen wir die Länge der Zeichenkette ab? Kann man das nicht umgehen?

7.2 Die Ausgabe von Strings mit Schlußzeichen

Wir formen den String so um:

1200 43 B7 B3 9E A5 96 92 9D 2A

Darin ist 2A der ASCII für «*» (asterisk). Nun ist eine im wesentlichen zweiteilige Aufgabe zu lösen:

1. Beim zeichenweisen Lesen des Strings muß der Asteriskus erkannt werden.
2. Beim Erkennen muß der Programmablauf geändert werden.

A7.2.1 Sie kennen bereits die benötigten Befehle. Codieren Sie diesen Ansatz!

A7.2.2 Schreiben Sie das Programm so um, daß es als Subroutine benutzt werden könnte:
1. Es soll in 3000 beginnen.
2. Statt HL soll DE benutzt werden.
3. Das DE-Register soll in einem Hauptprogramm gesetzt werden.
4. Der String soll mit 0D (carriage return) enden. Sie können Ihre Subroutine mit folgenden Strings und folgendem Hauptprogramm testen:

```
P7.2.3
1200                          41 B8 B8 92 A4 0D
1206                          A9 B8 A1 9E 9E 96 0D
120D                          AA A1 B3 B7 A4 21 0D
2000  LD    DE,1200           11 00 12    ;Anf.-Adr. 1.Str.
2003  CALL  3000              CD 00 30
2006  LD    DE,1206           11 06 12    ;Anf.-Adr. 2.Str.
2009  CALL  3000              CD 00 30
200C  LD    DE,120D           11 0D 12    ;Anf.-Adr. 3.Str.
200F  CALL  3000              CD 00 30
2012  HALT                    76
```

Sicherlich sind Sie mit dem Ergebnis nicht voll zufrieden. Die fehlenden Zwischenräume (spaces oder auch blanks) können Sie auf zweierlei Weise einfügen:
1. Sie können mit CALL 000C die Monitorroutine PRINTS (PRINT Space) aufrufen oder
2. in die Strings an passenden Stellen spaces (Code 20) einfügen.

Man hätte natürlich auch unter Einführung der erforderlichen spaces alle drei Wörter zu einem einzigen String zusammenfügen können. Unser Ziel ist es jedoch, die drei Wörter untereinander zu schreiben.

7.3 Die Routinen LETNL, MSG und MSGNL

Die Monitorroutine LETNL (LET New Line, aufgerufen mit CALL 0006) bewirkt die Ausgabe einer neuen Zeile. Fügen Sie sie ins Hauptprogramm ein!

Ihre Routine L7.2.2 bewirkt im wesentlichen dasselbe wie die mit CALL 0015 aufzurufende Monitorroutine MSG (message: Botschaft). Ersetzen Sie in P7.2.3 jeweils CALL 3000 durch CALL 0015!

Es kommt häufig vor, daß man nach der Ausgabe eines Strings eine neue Zeile beginnen will. Wir können dabei jeweils eine Programmzeile sparen, indem wir im EBS eine Routine MSGNL bereitstellen:

```
P7.3.1
MSGNL    AFED  CALL 0015    CD 15 00        ;MSG
         AFF0  CALL 0006    CD 06 00        ;LETNL
         AFF3  RET          C9
```

Wie Sie im Sharp-Handbuch auf Seite 152 nachlesen können, sind während des Ablaufs von MSG alle Register geschützt, und während des Ablaufs von LETNL alle bis auf AF. Das erlaubt ein wiederholtes Aufrufen von MSG und MSGNL:

```
P7.3.2
1200                       4D 5A 2D 37 30 30 0D
2000  LD   DE,1200         11 00 12
2003  CALL AFED            CD ED AF     ;MSGNL
2006  CALL AFED            CD ED AF     ;MSGNL
2009  CALL AFED            CD ED AF     ;MSGNL
200C  CALL AFED            CD ED AF     ;MSGNL
200F  HALT                 76
```

A7.3.3 Vielleicht versuchen Sie ein kürzeres Programm, das 22mal VOGEL-BUCHVERLAG untereinanderschreibt.

8
Zwei komfortablere Programmabschlüsse

Es war schon ein Fortschritt, als wir unsere Programme mit JP 0000 beendeten. Die letzten Beispiele nötigten uns jedoch, wieder auf das grobe HALT zurückzugreifen. Aber können wir nach einer Rechnung nicht auch den Speicherauszug per Programm anfordern?

8.1 Die EBS-Routine DUMP12

```
P8.1.1
DUMP12   AFDF  LD    DE,AFE5       11 E5 AF ;Stringanf.
         AFE2  JP    0D29          C3 29 0D ;in Monitor
         AFE5  Anfangsadresse      31 32 30 30
         AFE9  Endadresse + 1      31 32 41 30
```

Wir überzeugen uns davon, daß NULL12, MSGNL und DUMP12 geladen sind und saven alle drei Routinen mit SAFDFAFFF0000, wieder unter dem Namen EBS. Sofort erheben sich zwei Fragen:
1. Was haben wir damit gewonnen?
2. Wie funktioniert das?

Zu 1: Tippen Sie doch bitte P5.3.1 noch einmal ein und lassen Sie es in 2010 mit JP AFDF enden. Probelauf? Wenn alles richtig eingegeben wurde, müßten Sie jetzt nach abgeschlossener Multiplikation von (1200) und (1201) – hatten Sie dort auch zwei gültige Faktoren eingegeben? – den Hexdump ab 1200 mit dem fertigen Produkt in 1202 sehen. Im anderen Fall müßten Sie genau überprüfen, ob Sie auch alles richtig eingegeben haben.

Zu 2: Wir setzten DE auf die Anfangsadresse eines 8 Bytes langen Strings, der die Anfangs- und die um eins erhöhte Endadresse des

gewünschten Speicherauszugs enthält. Die Monitorroutine DUMP, in die wir sonst mit dem D-Befehl hineinspringen, beginnt mit einer Decodierung dieser beiden Adressen und sorgt dann für die Ausgabe des Speicherauszugs. Sie finden das Listing dieser Routine auf Seite 195 des Sharp-Handbuchs (Querformat, links unten).

Eine genauere Untersuchung wäre im Augenblick noch verfrüht. Experimentieren Sie jedoch mit anderen Anfangs- und Endadressen! Probieren Sie auch die folgende Variante aus:

```
P8.1.2
DUMPxx   A000  LD   HL,Anfang   21 00 00
         A003  LD   DE,Ende+1   11 34 01
         A006  JP   0D37        C3 37 0D
```

Vielleicht versuchen Sie zu einem späteren Zeitpunkt, den Vorgang genauer zu erfassen.

8.2 Die EBS-Routine PAUSKY

Mehrere der letzten Programme mußten wir mit HALT beenden, und es störte wieder, daß danach der Computer hängt. Wandeln Sie L7.3.3 unten wie folgt ab:

```
P8.2.1
400D  CALL 001B      CD 1B 00     ;GETKY
4010  JR   Z,400D    28 FB
4012  JP   0000      C3 00 00
```

Das Programm läuft nun wie bisher ab und bleibt wie bei HALT stehen. Wenn Sie aber dann eine beliebige Taste drücken, springt der Computer an den Anfang des Monitorprogramms.

Wie funktioniert das? Die Monitorroutine GETKY (001B) fragt die Tastatur ab. Ist keine Taste gedrückt, enthält der Akku den Wert 00, und im F-Register wird das Z-Flag gesetzt: Bedingter relativer Rücksprung nach 400D (vgl. Abschnitt 5.4). Wird hingegen eine Taste gedrückt, enthält der Akku ihren ASCII-Wert und das Z-Flag wird gelöscht: kein Rücksprung und weiter zum Monitor.

Uns gefällt der Dreizeiler so gut, daß wir ihn ins EBS aufnehmen:

```
P8.2.2
        AFD4   5mal NOP        00 00 00 00 00
PAUSKY  AFD9   CALL 001B       CD 1B 00
        AFDC   JR   Z,AFD9     28 FB
        AFDE   RET             C9
```

Die fünf NOPs sollen Platz reservieren für eine spätere Ergänzung. – Wir saven mit SAFD4AFFF0000 und ändern L7.3.3 so ab:

```
P8.2.3
400D  CALL AFD9        CD D9 AF    ;PAUSKY
4010  JP   0000        C3 00 00
```

PAUSKY wird mit CALL aufgerufen, da es mit RET endet. Leider wird PAUSKY häufig von kurzen bzw. schnellen Programmen ignoriert. Ausweg: Entweder gehe man vom vollgeschriebenen Bildschirm aus, denn das Scrolling erfordert – hier so gewünscht – eine kleine Zeitspanne, oder man greife behelfsmäßig auf HALT zurück. Wir werden diesen Mangel bald (in Abschnitt 10.5) beheben.

```
Geheime Verschlußsache

5000   21 08 50    LD   HL,5008
5003   06 1B       LD   B,1B
5005   34          INC  (HL)
5006   23          INC  HL
5007   10 FB       DJNZ 5004
5009   10 10       DJNZ 501B
500B   4F          LD   C,A
500C   CC 14 FF    CALL Z,FF14
500F   17          RLA
5010   0B          DEC  BC
5011   15          DEC  D
5012   55          LD   D,L
5013   A5          AND  L
5014   91          SUB  C
5015   B7          OR   A
5016   1F          RRA
5017   52          LD   D,D
5018   9D          SBC  A,L
5019   A0          AND  B
501A   AD          XOR  L
501B   20 0C       JR   NZ,5029
501D   20 07       JR   NZ,5026
501F   4F          LD   C,A
5020   05          DEC  B
5021   1A          LD   A,(DE)
5022   34          INC  (HL)
5023   23          INC  HL
5024   10 FC       DJNZ 5022
5026   76          HALT

Sie können den Mittelteil aber auch ersetzen durch

5012   46          LD   B,(HL)
5013   91          SUB  A,C
5014   97          SUB  A,A
5015   91          SUB  A,C
5016   A5          AND  L
5017   B2          OR   D
5018   AF          XOR  A
5019   A5          AND  L
501A   A3          AND  E
```

9
Indizierte Adressierung

Die Z80-CPU besitzt zwei Indexregister: IX und IY. Ihre Besonderheiten werden am besten am Beispiel deutlich. Betrachten wir den Befehl *LD A,(IX+d)* und nehmen wir an, daß IX = 1203 und d = 05 sei; dann würde der Akku indirekt indiziert aus Adresse 1208 geladen werden. Für d gilt dasselbe wie für die Sprungweiten: 100 > d > 7F wird als negative Zahl gedeutet.

9.1 Sechzehnstellige Addition

Wir speichern die beiden Summanden nach 1200...1207 und 1208...120F ein und schicken das Ergebnis in die Zeile darunter, nach 1210...1217:

```
P9.1.1
2000  LD    IX,1207       DD 21 07 12
2004  LD    B,08          06 08
2006  SCF                 37
2007  CCF                 3F
2008  LD    A,(IX+00)     DD 7E 00
200B  ADC   A,(IX+08)     DD 8E 08
200E  NOP                 00          ;später DAA
200F  LD    (IX+10),A     DD 77 10
2012  DEC   IX            DD 2B
2014  DJNZ  2008          10 F2
2016  JP    AFDF          C3 DF AF    ;DUMP12
```

Versuchen Sie, den Programmablauf Schritt für Schritt nachzuvollziehen: Es wird zuerst das im Dump rechts oben stehende Byte in den Akku geholt, dann das darunterstehende hinzuaddiert und die Teilsumme noch eine Zeile tiefer darunter abgelegt. Anschließend wird IX

um 1 vermindert (dekrementiert) und der Vorgang je eine Doppelstelle weiter links siebenmal wiederholt. Beachten Sie auch, daß die Rechnung um der Einheitlichkeit willen sogleich mit ADC beginnt und deshalb – hier noch ein wenig umständlich – Carry zu Beginn gelöscht wurde. Suchen Sie im Handbuch den Befehl SBC A,(IX+d), und subtrahieren Sie ebenso! Vergessen Sie auch nicht, dezimal zu rechnen!

9.2 Eingriffe in den Bildspeicher (Video-RAM)

Wie wir aus dem Handbuch (vgl. Seiten 128 f.) wissen, entsprechen dem Bildschirm die Adressen D000 bis D3E7. Lädt man nun eine dieser Speicherstellen mit einer Zahl, erscheint das dieser Zahl entsprechende Symbol auf dem Bildschirm. Geben wir z. B. ein MD3E7 ⟨CR⟩, CA ⟨CR⟩, so erscheint auf dem Bildschirm ein Strichmännchen. Um zusätzliche Grafiksymbole unterbringen zu können, verwendet Sharp für die Belegung des Bildschirmspeichers nicht den ASCII, sondern einen eigenen Bildschirmcode (siehe Sharp-Handbuch S. 159). Wir finden dort insgesamt vier Männlein, und wenn wir sie so anordnen: CA, CC, CD, DB, ..., dann schlagen sie Rad. Das bringt uns auf eine Idee, die wir schrittweise verwirklichen wollen. Wir speichern die Anzeigecodes in der angegebenen zyklischen Reihenfolge in die letzten 40 Bytes vor dem Video-RAM und transferieren sie mit LD A, (IX–78) und LD (IX+78) ,A in den Bildspeicher und erhalten – wenigstens im Prinzip – einen bewegten Radschläger auf dem Bildschirm:

```
P9.2.1
CFC0  LD    IX,D050     DD 21 50 D0
CFC4  LD    B,28        06 28
CFC6  LD    A,(IX-78)   DD 7E 88
CFC9  LD    (IX+78),A   DD 77 78
CFCC  INC   IX          DD 23
CFCE  DJNZ  CFC6        10 F6
CFD0  HALT              76
```

Das Programm läuft, doch sollte einiges verbessert werden: 1. Es läuft etwa tausendmal zu schnell ab. 2. Das Momentbild bleibt stehen, so daß ein Bildstreifen entsteht. Das jeweils letzte Bild müßte gelöscht werden.

10
Vom Prozessorstapel (Stack)

Ein bestimmter Speicherbereich – beim MZ-700 von 10EF bis 1000 (rückwärts numeriert!) – dient dem Prozessor als unmittelbarer Zwischenspeicher. Wir haben ihn schon benutzt, ohne ihn zu kennen.

10.1 Der Stapel als Rücksprunggedächtnis

Wenn man mit CALL eine Subroutine aufruft, dann legt der Prozessor die Adresse, zu der er nach Abarbeiten der Subroutine zurückkehren muß, auf dem Stapel ab. Eine Adresse ist 16 bit breit und füllt also zwei Bytes. Dementsprechend werden bei CALL jedesmal zwei Bytes auf dem Stack abgelegt und bei RET wieder heruntergeholt. Es gibt im Prozessor noch weitere, bisher unerwähnte Register, z. B. den Stackpointer (SP), der die jeweils aktuelle Adresse zum Stack enthält. Beim Ablegen auf dem Stapel wird SP um insgesamt 2 vermindert, beim Zurückholen vom Stapel um 2 erhöht. Nochmals: Der Stackpointer (SP, Zeiger auf die aktuelle Stackadresse) ist ein Register im Prozessor. Der Stapel selbst wird irgendwo im RAM organisiert. Weshalb nicht im ROM? Nun, dann könnte man ja nichts hineinschreiben.

10.2 Der Stapel als kurzzeitiger Datenspeicher

Man kann den Stapel aber auch willkürlich als kurzzeitigen Datenspeicher benutzen. Mit *PUSH* wird der Inhalt eines 16-bit-Registers (BC, DE, HL, AF, IX oder IY) auf dem Stapel abgelegt und mit *POP* wieder zurückgeholt. Man muß dabei sehr genau darauf achten, daß man insgesamt PUSH und POP gleich viele Male, in der richtigen Reihenfolge und nicht von anderen Stackoperationen gestört zur Ausführung

bringt. Sonst würde der Prozessor beim Zugriff auf den Stapel unsinnige Werte oder unsinnige Adressen vorfinden und der Computer «abstürzen».

A10.2.1 Speichern Sie nach 1200 12 und nach 1201 34 ein und schreiben Sie ein Programm, das diese zwei Bytes über DE zum Stapel transportiert und von dort über BC nach 1202 und 1203!

A10.2.2 Speichern Sie ab 1200 12 34 56 78 ein und schreiben Sie ein Programm, das die Inhalte von 1200 und 1201 gegen die Inhalte von 1202 und 1203 tauscht, indem der Inhalt von BC (kürzer: (BC)) einmal auf dem Stapel zwischengespeichert wird.

10.3 Stapelmanipulationen

Wir schreiben zunächst ein Trivialprogramm aus verschachtelten Subroutinen mit folgender Struktur:

```
0. Ebene                                              0. Ebene
              1. Ebene                      1. Ebene
                            2. Ebene
```

Es soll nicht mehr tun, als sich aus jeder Ebene mit dem jeweiligen Niveau zu melden. (Haben Sie auch das EBS geladen?)

```
P10.3.1
1200  '0.EBENE'        30 2E 45 42 45 4E 45 0D
1208  '1.EBENE'        31 2E 45 42 45 4E 45 0D
1210  '2.EBENE'        32 2E 45 42 45 4E 45 0D
2000  LD   DE,1200     11 00 12
2003  CALL MSGNL       CD ED AF
2006  CALL 2015        CD 15 20     ;zur 1.Ebene
2009  LD   DE,1200     11 00 12
200C  CALL MSGNL       CD ED AF
200F  CALL PAUSKY      CD D9 AF     ;oder HALT
2012  JP   MONITR      C3 00 00
2015  LD   DE,1208     11 08 12
2018  CALL MSGNL       CD ED AF
201B  CALL 2025        CD 25 20     ;zur 2.Ebene
201E  LD   DE,1208     11 08 12
2021  CALL MSGNL       CD ED AF
2024  RET              C9
```

```
2025  LD    DE,1210        11 10 12
2028  CALL  MSGNL          CD ED AF
202B  RET                  C9
```

(Hoffentlich empfinden auch Sie das allmähliche Einstreuen von Elementen der Assemblersprache als Lesehilfe!)

Das Programm zeigt ein verblüffendes Ablaufverhalten. Ruft man es z. B. nach D1200 auf, läuft es ganz nach Planung ab. Startet man es aber vom fast leeren Bildschirm aus, taucht für einen kurzen Moment die vorgesehene Schrift auf, um sogleich zu verschwinden. Beim systematischen Probieren findet man heraus, daß das Programm so schnell abläuft, daß man den Finger gar nicht schnell genug von der Tastatur lösen kann, so daß PAUSKY eine (noch) gedrückte Taste vorfindet. Ist der Bildschirm hingegen mit Zeichen angefüllt, wie z. B. nach D1200, so muß bei jeder neuen Schriftzeile gescrollt werden. Das verbraucht genügend Zeit, damit man den Finger bis zum Erreichen von PAUSKY wieder heben kann. Wir hielten vor PAUSKY 5 Bytes frei, um später eine Verzögerung davorzusetzen.

Wir beginnen nun mit der eigentlichen Stack-Manipulation. In 202B verwerfen wir eine Rückkehradresse, indem wir ein nicht weiter ausgewertetes POP DE einschieben:

```
P10.3.2
202B  POP   DE        D1
202C  RET             C9
```

Jetzt springt das Programm nach RET aus der 2. Ebene sofort in die 0. Ebene zurück. Das ist richtig, denn wir haben ja mit POP eine Rücksprungadresse verworfen.

Der Stapel hat eine LIFO-Struktur (last in first out). Das bedeutet, daß das zuerst Abgelegte als letztes wieder herauskommt. Vergleichen Sie das mit einem Tellerstapel: Der zuletzt aufgelegte Teller wird als erster abgenommen, der zuerst hingelegte (unterste) als letzter. Der Stapel wird von den höheren zu den niedrigeren Adressen hin belegt und in umgekehrter Richtung ausgelesen, wobei jeder Eintrag 2 Bytes umfaßt. Mit POP nahmen wir zwei Teller vom Stapel, jedoch ohne sie zu benutzen.

Wir führen nun bei gleicher Wirkung das Programm auf einem anderen Wege aus der 2. gleich in die 0. Ebene zurück, indem wir die

Stapelzugriffadresse, also den Inhalt des Stackpointers, um 2 erhöhen, d. h. zweimal inkrementieren.

```
P10.3.3
202B  INC  SP          33
202C  INC  SP          33
202D  RET              C9
```

Diese Variante führt zum gleichen Ablauf wie P10.3.2

A10.3.4 Was wird geschehen, wenn man statt dessen SP zweimal dekrementiert? Suchen Sie den Opcode, ändern Sie das Programm ab und deuten Sie Ihre Beobachtung!

A10.3.5 Benutzen Sie die neuen Befehle, um die Inhalte von H und L zu vertauschen. Fünf Stackoperationen genügen.

Eine weitere Manipulation: Unser Programm benutzt das HL-Register nicht. Wir erzwingen nun über dieses einen Sprung aus der 2. Ebene nach 2025, also in die 2. Ebene, indem wir diese Rücksprungadresse nach HL laden und mittels Stackpointer indirekt in den Stapel ablegen:

```
P10.3.6
202B  LD   HL,2025     21 25 20
202E  EX   (SP),HL     E3
202F  RET              C9
```

Der Befehl *EX (SP),HL* vertauscht (exchange) die Inhalte von HL und der in SP eingeschriebenen Adresse, und die gehört zum Stapel. – Wir sind mit dem Erfolg zufrieden, auch wenn sich nach einer Reihe planmäßiger Wiederholungen schließlich auch unplanmäßige Einschübe einstellen, weil wir den Stapel unterlaufen. Sie könnten versuchen, das Schritt für Schritt und Schleife für Schleife nachzuvollziehen.

A10.3.7 Bevor wir diesen Abschnitt verlassen, sollten Sie P10.3.1 noch ein wenig straffen.

10.4 Wir richten einen eigenen Stapel ein

Der Z80-Stapel liegt nicht zwangsläufig im Bereich 10EF...1000, sondern wurde vom Monitor softwaremäßig dort eingerichtet. Lesen Sie auf Seite 170 des Sharp-Handbuchs nach: Das Monitorprogramm beginnt mit einem Sprung nach 004A, und dort steht LD SP,10F0. Hier wird der Stackpointer so gesetzt, daß der erste Eintrag in 10EF erfolgt. Wir benutzen denselben Befehl und richten uns einen eigenen Stapel («Anwenderstapel») ab 1FFF ein:

```
P10.4.1
2000  LD   SP,2000      31 00 20
2003  CALL 2006         CD 06 20
2006  CALL 2009         CD 09 20
2009  CALL 200C         CD 0C 20
200C  CALL 200F         CD 0F 20
200F  JP   MONITR       C3 00 00
```

Dieses an sich sinnlose Programm richtet einen eigenen Stapel ein, schreibt einige Rücksprungadressen hinein, die Sie bitte mit D1FE0 kontrollieren wollen, und springt dann in den Monitor, der sofort wieder den «Systemstapel» einrichtet.

A10.4.2 Schreiben Sie ein Programm, das ab 1FFF (in Stapelrichtung zu lesen!) 12 34 12 35 12 36...13 33 einschreibt!

Wir lassen noch ein Stapelexperiment folgen, bei dem der momentane Stand des Stapelzeigers nach 1200 (+1) gerettet und dann von dorther wiederhergestellt wird:

```
P10.4.3
2040  CALL 2048         CD 48 20
2043  CALL BELL         CD 3E 00
2046  JR   2043         18 FB
2048  LD   (1200),SP    ED 73 00 12
204C  LD   SP,2000      31 00 20
204F  CALL 2052         CD 52 20
2052  CALL 2055         CD 55 20
2055  CALL 2058         CD 58 20
2058  LD   SP,(1200)    ED 7B 00 12
205C  RET               C9
```

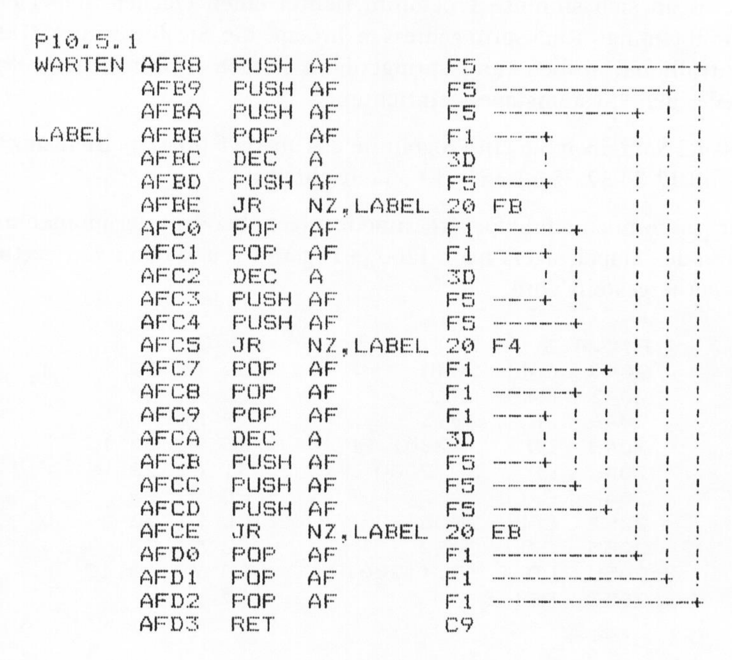
Das Dauerpiepsen nach Durchlauf zeigt uns, daß der Computer nicht irgendwo ausgestiegen ist, sondern genau die beabsichtigte Stelle erreichte.

10.5 Eine Verzögerungsroutine fürs EBS

Ihre Wirkung beruht darin, daß wir den Prozessor in drei Ebenen verschachtelt auf der Stelle zählen lassen. Schematisch sieht das so aus:

3 2 1 0 2 1 0 1 0 0 2 1 0 1 0 0 1 0 0 0
3 3 3 3 2 2 2 1 1 0 2 2 2 1 1 0 1 1 0 0
3 3 3 3 3 3 3 3 3 3 2 2 2 2 2 2 1 1 1 0

Dies sind 20 Zählschritte, lächerlich für einen Mikroprozessor. Wenn Sie Zeit haben, probieren Sie es doch einmal mit 4 4 4 aus! Sie werden dann schon glauben, daß auch eine CPU spürbar Zeit benötigt, um etwa mit A0 A0 A0 fertig zu werden. Wir schreiben die entsprechende Routine sogleich passend fürs EBS:



```
P10.5.1
WARTEN AFB8  PUSH AF           F5
       AFB9  PUSH AF           F5
       AFBA  PUSH AF           F5
LABEL  AFBB  POP  AF           F1
       AFBC  DEC  A            3D
       AFBD  PUSH AF           F5
       AFBE  JR   NZ,LABEL     20 FB
       AFC0  POP  AF           F1
       AFC1  POP  AF           F1
       AFC2  DEC  A            3D
       AFC3  PUSH AF           F5
       AFC4  PUSH AF           F5
       AFC5  JR   NZ,LABEL     20 F4
       AFC7  POP  AF           F1
       AFC8  POP  AF           F1
       AFC9  POP  AF           F1
       AFCA  DEC  A            3D
       AFCB  PUSH AF           F5
       AFCC  PUSH AF           F5
       AFCD  PUSH AF           F5
       AFCE  JR   NZ,LABEL     20 EB
       AFD0  POP  AF           F1
       AFD1  POP  AF           F1
       AFD2  POP  AF           F1
       AFD3  RET               C9
```

Die bedingten (Rück-)Sprünge werden korrekt ausgeführt, weil bei Ausführung der Befehle DEC 8-bit-Register (dekrementiere, siehe Z80-Handbuch) das Zero-Flag dem im angesprochenen Register stehenden Wert entsprechend gesetzt oder gelöscht wird. Ein CP 00 erübrigt sich deshalb.

Die eckigen Klammern am rechten Rand sollen zeigen, daß jedem PUSH genau ein POP entspricht. Das allein genügt nicht: Auch im Programmablauf müssen die Anzahlen der PUSHs und der POPs übereinstimmen! Das Benutzen der neuen Routine setzt voraus, daß unmittelbar vor ihrem Aufruf der Akku mit einem vernünftigen Wert (delay, Verzögerung) geladen wird.

Wir testen die neue Routine mit dem Kurzprogramm

```
P10.5.2
2000  LD    A,A0          3E A0
2002  CALL  AFB8          CD B8 AF
2005  JP    MONITR        C3 00 00
```

A10.5.3 Stellen Sie eine Tabelle auf, die die Verzögerung (in Sekunden) in Abhängigkeit von (A) angibt.

Wir benutzen die neue Routine sogleich zur Vervollständigung von PAUSKY:

```
P10.5.4
PAUSWK AFD4  LD    A,30          3E 30
       AFD6  CALL  WARTEN        CD B8 AF
```

10.6 Verbesserung des Radschlägers

Wir beziehen uns auf die Kritik an P9.2.1:

1. Wir bauen eine Verzögerungsroutine ein und probieren eine angemessene Verzögerung aus.

2. Sobald das neue Bild gezeichnet wird, löschen wir das alte; der Bildschirmcode für Space ist 00.

3. Im Verhältnis zu unseren kurzen Programmen bietet der MZ-700 beinahe unendlich viel Platz. Wir verlegen daher unseren versteckten Zeichenvorrat aus der Bildschirmnähe nach 1300...1327. Dann wird allerdings die Entfernung zwischen CFD8 und 1300 viel größer, als das

Offset d in (IX+d) erlaubt. Das ist nicht schlimm; wir nehmen einfach beide Indexregister, IX und IY, zu Hilfe:

```
P10.6.1
2000  LD    B,28          06 28
2002  LD    IX,1300       DD 21 00 13
2006  LD    IY,D3C0       FD 21 C0 D3
200A  LD    A,(IX+00)     DD 7E 00
200D  LD   (IY+00),A      FD 77 00
2010  LD   A,60           3E 60          ;variieren
2012  CALL WARTEN         CD B8 AF
2015  LD    (IY+00),00    FD 36 00 00    ;space
2019  INC   IX            DD 23
201B  INC   IY            FD 23
201D  DJNZ 200A           10 EB
201F  CALL PAUSKY         CD D9 AF
2022  JP    MONITR        C3 00 00
```

A10.6.2 Verändern und ergänzen Sie das Programm so, daß sich das Männlein vorwärts und rückwärts bewegt!

Unser Lösungsvorschlag L10.6.2 führte den Programmablauf in eine «endlose Schleife». Das wäre in der kommerziellen Datenverarbeitung ein ganz schwerer Fehler! Da der Homecomputer jedoch uns ganz allein zur Verfügung steht, nahmen wir den gewaltsamen Programmabbruch mit RESET in Kauf.

Eine Verbesserungsmöglichkeit bietet die Monitorroutine BRKEY (001E). Drückt man die Tasten SHIFT und BREAK gleichzeitig, kehrt der Prozessor aus dieser Routine mit gesetztem Zero-Flag zurück. Wir können das zur Unterbrechung der endlosen Schleife verwenden:

```
P10.6.3
2036  CALL BRKEY          CD 1E 00
2039  JR    NZ,2000       20 C5
203B  JP    MONITR        C3 00 00
```

Dieser Schluß ist so brauchbar, daß wir ihn zu einer EBS-Routine umschreiben:

```
P10.6.4
SBMONI   AFB3  CALL BRKEY   CD 1E 00
         AFB6  RET  NZ      C0
         AFB7  RST  00      C7
```

Freundliche Erinnerung: SAFB3AFFF0000 – EBS

Gleich zwei neue Befehle auf einmal? Fast könnte man antworten: nein!

RET NZ Dieses bedingte RET müßten Sie eigentlich jetzt schon unmittelbar, zumindest aber mit Hilfe des Z80-Handbuchs verstehen können.

RST 00 Vielleicht sehen Sie sich mit D000 noch einmal den Anfang des Monitorprogramms an. Es beginnt mit einer Reihe von Sprungbefehlen von der Form JP nn. Einige von ihnen können Sie mit Restart-Befehlen erreichen, z. B.

0000	JP 004A	mit	RST 00	=	C7
0018	JP 08A1	mit	RST 18	=	DF
0030	JP 01C7	mit	RST 30	=	F7
0038	JP 1038	mit	RST 38	=	FF

Die RST-Befehle entsprechen einem CALL. Näheres finden Sie im Z80-Handbuch.

Wir bauen eine neue Routine in den Schluß von P10.6.3 ein:

```
P10.6.5
2036  CALL SBMONI          CD B3 AF
2039  JR   2000            18 C5
```

Anregung: Lassen Sie doch P10.4.1 mit RST 00 enden und überzeugen Sie sich, daß die virtuelle Rücksprungadresse 2010 in «unseren» Stapel eingeschrieben wurde.

```
Musterstapelei

3000  LD   SP,D3E8    31 E8 D3
3003  LD   HL,F0F0    21 F0 F0
3006  LD   DE,0C09    11 09 0C
3009  AND  A          A7
300A  LD   B,A7       06 A7
300C  PUSH HL         E5
300D  INC  SP         33
300E  ADD  HL,DE      19
300F  PUSH HL         E5
3010  SBC  HL,DE      ED 52
3012  PUSH HL         E5
3013  INC  SP         33
3014  ADD  HL,DE      19
3015  PUSH HL         E5
3016  SBC  HL,DE      ED 52
3018  DJNZ 300C       10 F2
301A  HALT            76

Erfinden Sie weitere Muster!
```

11
Rotieren und Shiften

Man kann den Inhalt eines 8-bit-Registers oder einer Speicherstelle um ein Bit nach links oder rechts verschieben. In diesen Vorgang kann auch das Carry-Bit einbezogen werden. Wird dabei in einem Kreise (von 8 oder 9 Bits) herumgeschoben, heißt die Bewegung «Rotieren», anderenfalls «Shiften».

11.1 Ziel: Bitweise Multiplikation

Auch wenn wir bald ein weniger aufwendiges Verfahren kennenlernen werden, multiplizieren wir zunächst einmal ganz fachgerecht bitweise. Das Bit-Einmaleins wäre ein wunderbares Objekt für Grundschüler, denn sie brauchten sich nur vier Produkte einzuprägen: 0 ⋆ 0 = 0, 0 ⋆ 1 = 0, 1 ⋆ 0 = 0 und 1 ⋆ 1 = 1. Statt langer Erklärungen ein Beispiel:

```
19d * 23d = 10011 * 10111
                ---------------
                10111
               10111
              00000
             00000
            10111
            -------------
            110110101        = 437d
```

Bild 11.1 Bitweise Multiplikation

Das schwierigste hieran ist sicherlich nicht die Multiplikation, eher schon die Addition! Doch lassen Sie uns wieder schrittweise vorgehen:

11.2 Die Befehle *SLA (HL)* und *RL (HL)*

Wir speichern in 1200...1203 12 34 56 78 ein und wollen diese Zahlen um ein Bit nach links verschieben. Dazu stehen uns neben zahlreichen ähnlichen (siehe Z80-Handbuch) folgende Befehle zur Verfügung:
SLA (HL) CB 26 (shift left arithmetical) verschiebt den Inhalt der indirekt adressierten Speicherstelle um ein Bit nach links. In das 0. Bit (ganz rechts) wird eine 0 nachgeschoben; das 7. Bit (ganz links) wird ins Carry-Bit geschoben.
RL (HL) CB 16 (rotate left) bewegt die Bits in gleicher Weise, schiebt jedoch den momentanen Carry-Inhalt nach Bit 0 und holt Bit 7 ins Carry. Damit nimmt unser Programm folgende Gestalt an:

```
P11.2.1
1200                          12 34 56 78
2000   LD    HL,1203          21 03 12
2003   SLA   (HL)             CB 26
2005   DEC   HL               2B
2006   RL    (HL)             CB 16
2008   DEC   HL               2B
2009   RL    (HL)             CB 16
200B   DEC   HL               2B
200C   RL    (HL)             CB 16
200E   JP    DUMP12           C3 DF AF
```

Was geschieht? 1200 12 34 56 78
wird zu 1200 24 68 AC F0.

A11.2.2 Rufen Sie dieses Programm mehrmals auf und deuten Sie, was hier geschieht!

A11.2.3 Schreiben sie ab 2020 ein Programm, das ganz entsprechend den Speicherbereich 1200...1203 um ein Bit nach rechts shiftet. Suchen Sie die betreffenden Befehle im Z80-Handbuch. Experimentieren Sie mit J2020 und J2000!

11.3 Schieben in Breite von 8 Bytes

Wir bereiten eine Routine vor, die die ganze Datenzeile 1200...1207 um ein Bit nach links shiftet. Wir wollen diese Routine später anwenden und «retten» vorsichtshalber die benutzten Register HL und BC auf den Stapel, damit – falls benötigt – der momentane Wert dieser Register nach Rückkehr aus der Routine wiederhergestellt ist.

```
P11.3.1
2100  PUSH HL        E5
2101  PUSH BC        C5
2102  LD   HL,1207   21 07 12
2105  LD   B,08      06 08
2107  SCF            37
2108  CCF            3F
2109  RL   (HL)      CB 16
210B  DEC  HL        2B
210C  DJNZ 2109      10 FB
210E  POP  BC        C1
210F  POP  HL        E1
2110  RET            C9
```

Wodurch wurde hier das einsame SLA umgangen und dadurch das Programm vereinheitlicht?

A11.3.2 Schreiben Sie ab 2120 die nach rechts schiebende Variante.

11.4 Vervollständigung der Multiplikation

Lassen Sie uns weiterplanen mit dem Ziel, zwei (bis zu) achtstellige Zahlen miteinander zu multiplizieren. Dabei sind maximal sechzehnstellige Ergebnisse (also 8 Bytes) zu erwarten. Wir bezeichnen die Zeilen des Hexdumps wie folgt:

1200... Zeile1 enthält rechtsbündig Faktor1
1208... Zeile2 enthält rechtsbündig Faktor2
1210... Zeile3 enthält das fertige Produkt
1218... Zeile4 enthält Nullen (wegen Lesbarkeit)
1220... Zeile5 wird Schieberegister für Bitprüfung
1228... Zeile6 dient zum Shiften von Faktor 2

Faktor1 muß zunächst zur Bitprüfung nach Zeile5 und Faktor2 zum Linksshiften nach Zeile6 umgespeichert werden. Gleichzeitig wird Zeile3 auf null gesetzt, damit hier die einzelnen Zwischenprodukte aufaddiert werden können, und Zeile4 um der besseren Lesbarkeit willen. Das alles bewirkt

```
P11.4.1
2140  PUSH BC            C5
2141  LD   IX,1200       DD 21 00 12
2145  LD   B,10          06 10
2147  LD   A,(IX+00)     DD 7E 00
214A  LD   (IX+20),A     DD 77 20
214D  LD   (IX+10),00    DD 36 10 00
2151  INC  IX            DD 23
2153  DJNZ 2147          10 F2
2155  POP  BC            C1
2156  RET                C9
```

Als nächstes benötigen wir eine Routine, die von rechts angefangen jeweils 1 Byte von Zeile6 und ein Byte von Zeile3 addiert und die jeweilige Teilsumme wieder nach Zeile3 ablegt:

```
P11.4.2
2160  PUSH BC            C5
2161  LD   B,08          06 08
2163  LD   IX,1217       DD 21 17 12
2167  SCF                37
2168  CCF                3F
2169  LD   A,(IX+18)     DD 7E 18
216C  ADC  A,(IX+00)     DD 8E 00
216F  LD   (IX+00),A     DD 77 00
2172  DEC  IX            DD 2B
2174  DJNZ 2169          10 F3
2176  POP  BC            C1
2177  RET                C9
```

Wir haben nun folgende Subroutinen bereitgestellt:
2100 shiftet die Zeile1 um 1 Bit nach links. Damit künftig Zeile6 in dieser Weise geshiftet wird, muß (2103) in 2F geändert werden.
2120 shiftet die Zeile1 um 1 Bit nach rechts. Damit künftig Zeile5 in dieser Weise geshiftet wird, muß (2123) in 20 geändert werden.
2140 initialisiert die Rechenfelder und erwartet die Faktoren rechtsbündig in Zeile1 und Zeile2.

2160 addiert jeweils den nach links geshifteten Faktor 2 zur momentanen Summe der bisherigen Teilprodukte.

Wir können nun das Hauptprogramm bzw. die Koordinierung der Unterprogramme planen:
1. Manuelle Eingabe der beiden Faktoren in Zeile1 und Zeile2.
2. Initialisieren der Rechenfelder: Routine 2140.
3. Faktor1 (in Zeile5) nach rechts shiften und danach Carry abfragen, ob Teiladdition erfolgen muß: Routine 2120.
4. Wenn ja, dann Teiladdition durchführen: Routine 2160.
5. Faktor2 (in Zeile6) nach links shiften wie das Multiplikationsschema in Abschnitt 10.3 vorschreibt: Routine 2100.
6. Das Ganze (ab 3.) 32mal durchführen, weil die Multiplikation über eine Breite von 8 Nibbles = 32 bit durchzuführen ist.

Damit nimmt unser Hauptprogramm Gestalt an:

```
P11.4.3
2000  CALL 2140      CD 40 21
2003  LD   B,20      06 20
2005  CALL 2120      CD 20 21
2008  JR   NC,200D   30 03
200A  CALL 2160      CD 60 21
200D  CALL 2100      CD 00 21
2010  DJNZ 2005      10 F3
2012  JP   DUMP12    C3 DF AF
```

Wir haben damit ein schon ziemlich kompliziertes Gebilde geschaffen und sollten es sorgfältig testen. Einfache Beispiele wären:

00 00 00 00 00 00 00 05
00 00 00 00 00 00 00 07
00 00 00 00 00 00 00 23h = 35d

oder

00 00 00 00 11 11 11 11
00 00 00 00 11 11 11 11
01 23 45 67 87 65 43 21.

Wir haben dieses Programm in Abschnitten geschrieben, um an und aus seiner modularen Struktur zu lernen. Vielleicht möchten Sie die Teile enger zusammenfügen?

A11.4.4 Sind eigentlich alle PUSHs und POPs wirklich nötig?

12
Ein wenig Boolesche Algebra – oder auch angewandte Mengenlehre

Einige Grundbegriffe der Booleschen Algebra haben in etwas veränderter Form Einzug in die Maschinensprache gehalten. Aus der Mengenlehre stammt der Begriff «Schnittmenge», auch «Durchschnitt» genannt. Zur Schnittmenge gehören diejenigen Elemente zweier Mengen, die in beiden Mengen enthalten sind. Fassen wir also einmal achtstellige Binärzahlen als Mengen auf, die maximal einen 128er, einen 64er, einen 32er, ... und einen 1er enthalten.

12.1 Der Befehl *AND* bildet eine Schnittmenge

Wenn wir nun in diesem Sinn die Schnittmenge von zwei Einbytezahlen bilden, kann das so aussehen:

Erste Zahl:	0010 1101	hexadezimal 2D
Zweite Zahl:	0101 1011	hexadezimal 5B
Schnittmenge:	0000 1001	hexadezimal 09

Der entsprechende Maschinenbefehl heißt AND: Die beiden Elemente 8er UND 1er sind in beiden Zahlen (Mengen) 2D und 5B enthalten. Probieren Sie folgendes kurzes Programm aus und machen Sie sich mit seiner Hilfe mit dem Befehl AND vertraut. Dabei sollen die beiden vorgegebenen Zahlen (Operanden) aus 1200 und 1201 geholt und das Ergebnis in 1202 abgelegt werden.

```
P12.1.1
2000   LD    HL,1200       21 00 12
2003   LD    A,(HL)        7E
2004   INC   HL            23
2005   AND   (HL)          A6
2006   INC   HL            23
```

```
2007  LD   (HL),A     77
2008  JP   DUMP12     C3 DF AF
```

Beispiele: 2D AND 5B = 09, 7B AND 62 = 62. Denken Sie sich weitere aus und deuten Sie jedes bitweise.
Anwendungen: Man nennt AND-Operationen manchmal «Maskieren». Durch «Undieren» mit einer «Maske» kann man bestimmte Informationen über eine Zahl erhalten.

A12.1.2 Suchen Sie geeignete Masken für folgende Feststellungen:

a) Ist die Zahl gerade oder ungerade?
b) Welchen Rest läßt sie nach Division durch 4, 8, 16, 32, 64, 128?
c) Ist sie > 127 bzw. negativ?

Auf eine Besonderheit des Befehls *AND A* sei hingewiesen: Hier wird der Inhalt des Akkus mit sich selbst undiert. Sein Wert ändert sich dabei nicht, aber es geschehen zwei willkommene Dinge:
1. Carry wird gelöscht. Das ersetzt SCF mit nachfolgendem CCF.
2. Z-Flag wird je nach (A) gesetzt oder gelöscht.

12.2 Befehle *OR* und *XOR*

Gerd sagte: «Wenn Hans oder Franz kommt, gehen wir zum Schwimmen». Dieser Satz läßt zwei verschiedene Deutungen zu:

OR Das einschließende Oder läßt auch zu, daß beide kommen: Gerd würde auch dann zum Schwimmen gehen.
XOR Das eXklusive Oder würde diesen Fall ausschließen. Wenn beide kämen, würde Gerd nicht zum Schwimmen gehen.

Der Prozessor wendet diese Funktionen allerdings nicht auf sprachlich formulierte Sätze an, sondern auf Binärzahlen, wie bei AND erklärt, und mit folgenden Ergebnissen:

```
     0100 1001          0100 1001
  OR 0111 0011      XOR 0111 0011
   = 0111 1011        = 0011 1010
```

Probieren Sie die beiden neuen Funktionen mittels P12.1.1 reichlich aus. Suchen Sie die Opcodes von OR (HL) und XOR (HL) im Z80-Handbuch!

A12.2.1 Schreiben Sie ein Programm, das das MSB einer in 1200 stehenden Zahl mit nachfolgender Null in 1201 ablegt und das LSB mit führender Null nach 1202.

A12.2.2 Wenn der Akku mit 41 geladen ist, bewirkt CALL PRNT die Ausgabe eines «A». Es soll jedoch «41» ausgegeben werden. Schreiben Sie ein Programm, das eine in 1200 stehende Dezimalzahl auf dem Bildschirm ausgibt!

A12.2.3 Erweitern Sie L12.2.3 so, daß auch Hexadezimalzahlen ausgegeben werden können!

```
        Kleine Xorrerei

KLXORI  3000   CD 13 30    CALL XORINF
        3003   CD 1F 30    CALL XORCOL
        3006   CD 1F 30    CALL XORCOL
        3009   CD 13 30    CALL XORINF
        300C   CD 1F 30    CALL XORCOL
        300F   CD 1F 30    CALL XORCOL
        3012   76          HALT
XORINF  3013   21 00 D0    LD   HL,D000
        3016   06 19       LD   B,19
LABEL1  3018   CD 31 30    CALL FLASH
        301B   23          INC  HL
        301C   10 FA       DJNZ LABEL1
        301E   C9          RET
XORCOL  301F   21 00 D8    LD   HL,D800
        3022   06 19       LD   B,19
LABEL2  3024   3E 77       LD   A,77
        3026   AE          XOR  (HL)
        3027   77          LD   (HL),A
        3028   3E 20       LD   A,20
        302A   CD B8 AF    CALL WARTEN
        302D   23          INC  HL
        302E   10 F4       DJNZ LABEL2
        3030   C9          RET
FLASH   3031   C5          PUSH BC
        3032   06 19       LD   B,19
LABEL3  3034   3E 10       LD   A,10
        3036   CD B8 AF    CALL WARTEN
        3039   3E FF       LD   A,FF
        303B   AE          XOR  (HL)
        303C   77          LD   (HL),A
        303D   10 F5       DJNZ LABEL3
        303F   C1          POP  BC
        3040   C9          RET

Starten Sie das unveränderte Programm nach RESET
oder nach J0000. Das EBS muß geladen sein!
```

13
Halbbytes-Swap

Die Befehle *RLD* bzw. *RRD* (rotate left (bzw. right) decimal)) lösen einen Ringtausch von MSB und LSB der mittels HL indirekt adressierten Speicherstelle und dem LSB des Akkus aus. Er heißt Halbbytes-Swap.

13.1 Dezimale Multiplikation

Das Multiplikationsprogramm von Kapitel 11 kann nicht zur dezimalen Multiplikation verwendet werden, weil nicht alle Bits gleichberechtigt geshiftet werden dürfen: Nibbles wie 1010, 1100 usw. dürfen nicht auftreten. An die Stelle des Shiftens und Rotierens tritt jetzt der Halbbytes-Swap.

A13.1.1 Schreiben Sie ein Programm, das den Inhalt von 1201 linksherum swappen läßt. Der anfängliche Akku-Inhalt soll in 1200 bereitgestellt werden. Ablage des Ergebnisses nach 1208 und 1209.

Wir folgen nun wieder dem modularen Aufbau von Kapitel 11:

A13.1.2 Schreiben Sie eine Routine, die Zeile6 um ein Nibble nach links swappt, von rechts eine Null nachschiebt und das höchstwertige Nibble verwirft. Beginnen Sie das Programm in 2100.

A13.1.3 Schreiben Sie ab 2120 eine Routine, die ganz entsprechend Zeile5 nach rechts swappt.

Wir stellen jetzt sicher, daß sich die Unterprogramme P11.4.1 (ab 2140) und P11.4.2 (ab 2160) wieder an ihrem Platz befinden, und gehen an die Entwicklung des Hauptprogramms:
1. Manuelle Eingabe der beiden Faktoren in Zeile1 und Zeile2.
2. Initialisieren der Rechenfelder: Routine 2140.
3. Faktor1 (in Zeile5) nach rechts swappen und Akku mit 0F maskieren,

damit nur das hinausgeschobene Nibble übrigbleibt. Es bestimmt, ob und wie oft addiert werden muß.
4. Wenn A = 0, dann Mehrfachaddition überspringen, sonst (A) nach B transferieren und (B)-mal addieren: Routine 2160.
5. Faktor2 (in Zeile6) nach links swappen: Routine 2100.
6. Das Ganze (ab 3.) achtmal, weil unsere Faktoren (bis zu) 8 Nibbles breit sind. Damit sieht unser Hauptprogramm so aus:

```
P13.1.4
2000  CALL 2140     CD 40 21   ;initialisieren
2003  LD   B,08     06 08
2005  CALL 2120     CD 20 21   ;nach rechts sh.
2008  AND  0F       E6 0F      ;Zero-Flag = ?
200A  JR   Z,2014   28 08
200C  PUSH BC       C5
200D  LD   B,A      47
200E  CALL 2160     CD 60 21   ;addieren
2011  DJNZ 200E     10 FB
2013  POP  BC       C1
2014  CALL 2100     CD 00 21   ;nach links sh.
2017  DJNZ 2005     10 EC
2019  JP   DUMP12   C3 DF AF
```

Probieren Sie wieder viele Beispiele aus; zwei zur Kontrolle:

1000 ⋆ 1000 = 01000000 (hex)
03E8 ⋆ 03E8 = 000F4240 (hex)

Wir können nun die Hexadezimalmultiplikation in eine Dezimalmultiplikation überführen, indem wir in die Additionsroutine 2160 ein DAA einfügen. Wir benutzen dabei ganz bewußt den Befehl AND A (vgl. Abschnitt 12.1) zum Löschen von Carry-Flag und stellen uns vor, die Additionsroutine stände oben und unten eng ins Programm eingeschlossen. Dann käme uns der Gewinn eines Bytes (27 statt 37 3F) sehr zustatten, denn wir hätten Platz für DAA, für das wir keinen Platzhalter (NOP) bereitgestellt hatten.

```
P13.1.5
....  .... .........    .. .. ..
2167  AND  A            A7
2168  LD   A,(IX+18)    DD 7E 18
216B  ADC  A,(IX+00)    DD 8E 00
216E  DAA               27
....  .... .........    .. .. ..
2174  DJNZ 2168         10 F2
....  .... .........    .. .. ..
```

14
Weitere Monitor- und EBS-Routinen zur Erhöhung des Ein-/Ausgabe-Komforts

Wir haben bisher stets die zu verarbeitenden Daten in unserem Arbeitsbereich ab 1200 abgelegt und danach das Verarbeitungsprogramm aufgerufen. Es erhebt sich die Frage, ob man die Daten nicht auch unter Programmkontrolle eingeben kann, so daß sich die Ausführung eines Programms vereinfacht.

14.1 Die Monitorroutine GETKY

Der hervorragend dokumentierte Monitor des MZ-700 bietet zwei komfortable Eingaberoutinen an, HLHEX und 2HEX. Lesen Sie auf jeden Fall die Kommentare dazu. Wir benutzen diese beiden Routinen hier nicht, sondern versuchen zu verstehen, wie derartiges vor sich geht, indem wir selber ähnliche Routinen aufstellen.

Wir hatten in P8.2.1 und PAUSKY bereits die Routine GETKY benutzt. Wir untersuchen sie jetzt näher:

```
P14.1.1
3000  CALL  NULL12       CD F4 AF
3003  LD    HL,1200      21 00 12
3006  CALL  GETKY        CD 1B 00
3009  LD    (HL),A       77
300A  INC   HL           23
300B  CALL  GETKY        CD 1B 00
300E  LD    (HL),A       77
300F  INC   HL           23
3010  CALL  GETKY        CD 1B 00
3013  LD    (HL),A       77
3014  JP    DUMP12       C3 DF AF
```

Das Ergebnis ist erstaunlich. Tippen wir J3000 ein und drücken wir nur

sehr kurz ⟨CR⟩, dann erhalten wir 00 00 00 .. .. . Drücken wir ⟨CR⟩ jedoch länger, erhalten wir bis zu dreimal 66. Das muß untersucht werden: Starten wir das Programm noch einmal und drücken dabei ⟨CR⟩ nur sehr kurz und sofort anschließend die Taste «A», dann kann es uns mit Geschick und Glück gelingen, auch einmal 66 41 41 .. .. zu erhalten. Vielleicht verlängern Sie das Programm noch ein wenig? Und dann erweitern wir es der Größe des Anzeigefelds entsprechend auf 160 Eingaben:

```
P14.1.2
3000  CALL NULL12      CD F4 AF
3003  LD   HL,1200     21 00 12
3006  LD   B,A0        06 A0
3008  CALL GETKY       CD 1B 00
300B  LD   (HL),A      77
300C  INC  HL          23
300D  DJNZ 3008        10 F9
300F  JP   DUMP12      C3 DF AF
```

Wenn wir nun J3000 eintippen und dann ohne besondere Hast nacheinander die Tasten CR, 1, 2, 3, 4, 5, .. drücken, zeigt der Hexdump je einige Male 66, 00, 31, 32, 33, 34, 35, .. an. Das verrät uns folgendes:

1. Die Tastatur wird vom Programm in mäßig schneller Folge mehrmals hintereinander abgefragt.
2. Eine gedrückte Taste wird dabei mehrfach registriert.
3. 66 ist die Codierung von CR
   00 ist die Codierung von «keine Taste»
   31 ist die Codierung von «1» usw.

Sehen Sie sich dazu den Kommentar zu GETKY im Sharp-Handbuch an.

14.2 Tastaturentprellung per Software

Wir sind damit an ein wichtiges Teilproblem der Tastaturabfrage gelangt: Ermittelt diese (festgestllt von GETKY) den Code 66, so könnte der noch von einem früheren CR stammen; neue Abfrage! Nach «CR» müßte zunächst einmal «keine Taste» festgestellt werden. Dann muß erneut abgefragt werden, bis eine «gültige» Taste gedrückt wird. Nun

haben Metallteile, die man schnell aneinanderpreßt, es an sich, zu «prellen», bis sie endlich ruhig einander berühren. In dieser Phase muß abgewartet werden, bis der schnelle Wechsel von Nullen und gültigen Werten abgeklungen ist. Wir ermitteln das, indem wir den jeweils ermittelten vielleicht gültigen Code in C laden und mit dem nächsten erhaltenen Code vergleichen. Bei Gleichheit nehmen wir an, jetzt den wirklich gültigen Code gefunden zu haben. Der Vorgang ist abgeschlossen, wenn «keine Taste» festgestellt wird.

Wir schreiben jetzt ein Programm, das achtmal die «Tastaturentprellung» softwaremäßig vollzieht und die Eingaben zu unserer Kontrolle und Deutung nach 1200...1207 abspeichert:

```
P14.2.1
3000  CALL  NULL12      CD F4 AF
3003  LD    HL,1200     21 00 12    ;Anfangsadresse
3006  LD    B,08        06 08       ;8 Durchläufe

3008  CALL  GETKY       CD 1B 00
300B  AND   A           A7          ;prüfen, ob 00
300C  JR    Z,3008      28 FA       ;wenn 00 zurück
300E  CP    66          FE 66       ;prüfen, ob CR
3010  JR    Z,3008      28 F6       ;wenn CR zurück

3012  LD    C,00        0E 00       ;Anfangswert
3014  CALL  GETKY       CD 1B 00
3017  CP    C           B9          ;A = C ?
3018  LD    C,A         4F          ;nach C zwi'sp.
3019  JR    NZ,3014     20 F9       ;wenn <> zurück
301B  LD    (HL),A      77          ;abspeichern
301C  INC   HL          23

301D  CALL  GETKY       CD 1B 00
3020  AND   A           A7          ;prüfen ob 00
3021  JR    NZ,301D     20 FA       ;wenn <> zurück
3023  DJNZ  3008        10 E3       ;neuer Durchlauf
3025  JP    DUMP12      C3 DF AF
```

Unser Programm arbeitet korrekt, auch wenn wir z. B. A, A, B, B, C, C, D, D eingeben. (Probieren Sie aus, ob beim MZ-700 die Feststellung der Konstanz der Tastaturabfrage nötig ist, indem Sie 3012...301A mit NOPs überschreiben.) Das Monitorprogramm enthält bereits eine entsprechende Routine; sie heißt ??KEY (09B3), und wir probieren sie sogleich aus:

```
P14.2.2
3030  CALL  NULL12      CD F4 AF
3033  LD    HL,1200     21 00 12
3036  LD    B,08        06 08
3038  CALL  ??KEY       CD B3 09
303B  LD    (HL),A      77
303C  INH   HL          23
303D  DJNZ  3038        10 F9
303F  JP    DUMP12      C3 DF AF
```

An der Routine ??KEY gefällt uns:
1. Sie liegt bereits fertig vor.
2. Der Cursor zeigt die Schreibposition an.

Für unsere Zwecke vermissen wir:
1. ??KEY liefert nicht den ASCII-, sondern den Anzeigecode.
2. Die Eingabe wird nicht angezeigt.

A14.2.3 Schreiben Sie P14.2.2 so um, daß es auch in diesen Punkten unseren Vorstellungen entspricht. Benutzen Sie dazu ?DACN (siehe Sharp-Handbuch Seite 155).

14.3 Die EBS-Routinen KYECH und CLS

L14.2.3 läuft einwandfrei. Geben Sie auch einmal ein 1, 2, ⟨CR⟩, 3, ... CR wird wieder als 66 codiert. Uns gefällt diese Kombination, so daß wir sie sogleich zu einer EBS-Routine KYECH umschreiben. KYECH soll heißen «Key mit Echo», wobei man unter «Echo» versteht, daß das eingegebene Zeichen auch auf dem Bildschirm ausgegeben wird.

```
P14.3.1
KYECH  AFA7  CALL ??KEY   CD B3 09
       AFAA  CALL ?DACN   CD CE 0B
       AFAD  PUSH AF      F5
       AFAE  CALL PRNT    CD 12 00
       AFB1  POP  AF      F1
       AFB2  RET          C9
```

Wir lassen sofort noch eine Routine CLS folgen, die im wesentlichen aus einem Steuerzeichen mit nachfolgendem PRNT besteht:

```
P14.3.2
CLS      AF9F   PUSH AF          F5
         AFA0   LD   A,16        3E 16
         AFA2   CALL PRNT        CA 12 00
         AFA5   POP  AF          F1
         AFA6   RET              C9
```

Wir saven mit SAF9FAFFF0000 – EBS und überprüfen das Erreichte mit einem Kurzprogramm:

```
P14.3.3
2000   CALL  NULL12              CD F4 AF
2003   CALL  CLS                 CD 9F AF
2006   CALL  KYECH               CD A7 AF
2009   LD    (1200),A            32 00 12
200C   JP    DUMP12              C3 DF AF
```

14.4 Die EBS-Routine INBYTE

Es kommt häufig vor, daß eine zweistellige Hexadezimalzahl, also ein Byte, einzugeben ist. Soll z. B. A5 eingegeben werden, so wären dazu die Tasten «A» und «5» zu drücken, und KYECH würde 41 und 35 als Eingaben registrieren. Diese beiden Teile sind nun wieder zu «A5» zusammenzusetzen. Wir tun das zunächst, ohne zu prüfen, ob bei der Eingabe eines jeden Nibbles ganz korrekt keine andere als eine der Tasten 0...9 bzw. A...F gedrückt wurde. Das Programm muß von folgender Struktur sein:

1. Eingabe der ersten Ziffer.
2. Wenn ASCII > 40, dann von 41 auf 4A, von 42 auf 4B, ... von 46 auf 4F, also um 9 erhöhen.
3. Um 4 bit nach links shiften und zwischenspeichern.
4. Eingabe der zweiten Ziffer.
5. wie 2.
6. Mit 0F maskieren.
7. Zum zwischengespeicherten Wert addieren.
8. Abspeichern.

```
P14.4.1
3100  CALL  KYECH        CD A7 AF
3103  CALL  311E         CD 1E 31
3106  SLA   A            CB 27
3108  SLA   A            CB 27
310A  SLA   A            CB 27
310C  SLA   A            CB 27
310E  LD    B,A          47
310F  CALL  KYECH        CD A7 AF
3112  CALL  311E         CD 1E 31
3115  AND   0F           E6 0F
3117  ADD   A,B          80
3118  LD    (1200),A     32 00 12
311B  JP    DUMP12       C3 DF AF
311E  CP    40           FE 40
3120  RET   C            D8
3121  ADD   A,09         C6 09
3123  RET                C9
```

Bei aller Nützlichkeit weist diese Lösung noch den u. U. gefährlichen Mangel auf, daß auch falsche Eingaben wie z. B. «XY», «7G», «(!» usw. angenommen werden. Probieren Sie das bitte aus. Die Deutung ist gar nicht schwer.

Zur Erkennung derartiger Eingabefehler formulieren wir nun eine Routine INBYTE, die mittels KYECH zwei Nibbles von der Tastatur aufnimmt und jedes prüft, ob es zulässiger Bestandteil einer zweistelligen Hexadezimalzahl ist. Dazu wird das eingelesene Halbbyte mit einer Tabelle verglichen, die die zulässigen Codes 30...39 und 41...46 enthält. Zum Umwandeln dieser Codes in das jeweils aufzubauende Nibble beginnen wir im D-Register mit 00 und inkrementieren nach jedem erfolglosen Vergleich. Das zuerst ermittelte Halbbyte wird in C zwischengespeichert und viermal nach links geshiftet; anschließend wird das zweite ermittelt und addiert. Wurde eine Eingabe mit allen 16 Elementen der Tabelle ergebnislos verglichen, ist sie unzulässig. Drei Piepstöne zeigen das an, und der Cursor wird mittels Steuercode C4 und Monitorroutine ?DPCT (Seite 155 des Sharp-Handbuchs) auf das Echo der Fehleingabe zurückbewegt.

Wir verlegen INBYTE sofort an den richtigen Ort und verwenden von hier ab mehr als bisher Ausdrucksweisen der Assemblersprache, um Sie allmählich auch mit ihr bekanntzumachen. Die symbolischen Adressen – auch Labels genannt – sind Programmiervariable: Sie können z. B.

«EIN» in diesem Sinne als eine Variable mit dem Wert AF6F auffassen. Tippen Sie das Programm nicht einfach ab, sondern versuchen Sie, es anhand der gegebenen Erklärungen Schritt für Schritt zu verstehen.

```
P14.4.2
INBYTE   AF57  PUSH BC         C5
         AF58  PUSH DE         D5
         AF59  PUSH HL         E5
         AF5A  CALL EIN        CD 6F AF
         AF5D  LD   C,D        4A
         AF5E  SLA  C          CB 21
         AF60  SLA  C          CB 21
         AF62  SLA  C          CB 21
         AF64  SLA  C          CB 21
         AF66  CALL EIN        CD 6F AF
         AF69  LD   A,C        79
         AF6A  ADD  A,D        82
         AF6B  POP  HL         E1
         AF6C  POP  DE         D1
         AF6D  POP  BC         C1
         AF6E  RET             C9
EIN      AF6F  CALL KYECH      CD A7 AF
         AF72  LD   HL,TABLE   21 8F AF
         AF75  LD   B,10       06 10
         AF77  LD   D,00       16 00
LABEL    AF79  CP   (HL)       BE
         AF7A  RET  Z          C8
         AF7B  INC  HL         23
         AF7C  INC  D          14
         AF7D  DJNZ  LABEL     10 FA
         AF7F  CALL BELL       CD 3E 00
         AF82  CALL BELL       CD 3E 00
         AF85  CALL BELL       CD 3E 00
         AF88  LD   A,C4       3E C4     ;Cursor links
         AF8A  CALL ?DPCT      CD DC 0C
         AF8D  JR   EIN        18 E0
TABLE    AF8F                  30 31 32 33 34 35 36 37
         AF97                  38 39 41 42 43 44 45 46
```

Wir testen das Erreichte mit einem Kurzprogramm:

```
P14.4.3
TEST    2000  LD    B,08       06 08
        2002  LD    HL,1200    21 00 12
LABEL1  2005  CALL  INBYTE     CD 57 AF
        2008  LD    (HL),A     77
        2009  INC   HL         23
        200A  DJNZ  LABEL1     10 F9
        200C  JP    DUMP12     C3 DF AF
```

14.5 Die EBS-Routinen IN2HEX, IN4HEX und IN6HEX

In der Praxis sind häufig nicht zweistellige, sondern vierstellige und manchmal sogar sechsstellige Hexadezimalzahlen einzugeben und in von Neumannscher Anordnung abzulegen. Letzteres erreichen wir, indem wir die Zieladresse des MSB nach HL laden und das MSB dann indirekt nach dort ablegen. Danach wird HL dekrementiert, das LSB eingegeben und ebenfalls indirekt abgelegt.

A14.5.1 Schreiben Sie ein Programm nach diesen Angaben und ordnen Sie es wie immer vor dem EBS ein!

14.6 Die EBS-Routine OUTBYT

Wer A sagt, muß auch B sagen, und auf INBYTE folgt OUTBYT. Diese Routine soll ein im Akku stehendes Byte auf dem Bildschirm ausgeben. Dazu sind beide Nibbles in ASCIIs zu verwandeln. Das besorgt die Monitorroutine ASC (siehe Sharp-Handbuch).

A14.6.1 Entwerfen Sie eine EBS-Routine, die das leistet. Der Raum von AF30 bis AF46 genügt!

A14.6.2 Wir müssen OUTBYT auf Zuverlässigkeit untersuchen. Speichern Sie in 1200...1207 Zahlen ein, die mittels OUTBYT auf den Bildschirm geschrieben werden. Schreiben Sie ein geeignetes Kurzprogramm!

14.7 Die EBS-Routinen OUT2HX, OUT4HX und OUT6HX

A14.7.1 Schreiben Sie einen INnHEX entsprechenden Dreierblock, der ganz analog 3, 2 oder 1 Byte(s) ausgibt. Der Speicherraum von AF20 bis AF2F genügt!

A14.7.2 Schreiben Sie ein hübsches Kurzprogramm, das IN6HEX und OUT6HX in je einer fünfmal zu durchlaufenden Schleife testet. Legen Sie Wert auf gute Überschaubarkeit!

14.8 Die EBS-Routine OUTBIN

Wir bereiten jetzt eine Routine vor, die das im Akku stehende Byte als Binärzahl ausgibt. Wir gehen wieder schrittweise vor und holen uns eine Zahl aus 1200, von der wir zunächst nur die beiden ersten Bits ausgeben:

```
P14.8.1
2000  LD    A,(1200)    3A 00 12
2003  LD    B,00        06 00
2005  SLA   A           CB 27
2007  RL    B           CB 10
2009  SLA   B           CB 20
200B  SLA   B           CB 20
200D  SLA   B           CB 20
200F  SLA   A           CB 27
2011  RL    B           CB 10
2013  LD    A,B         78
2014  CALL  OUTBYT      CD 30 AF
2017  CALL  PAUSWK      CD D4 AF
201A  JP    DUMP12      C3 DF AF
```

A14.8.2 Bitte überlegen Sie:
a) Was bewirken die Befehle in 2005 und 2007?
b) Was bewirken die drei SLA Bs in 2009...200B?
c) Was geschieht in 200F?
d) Was geschieht in 2011?
e) Weshalb ist 2013 unbedingt nötig?

Hier folgt ein Programm, das in endloser Folge zweistellige Hexadezimalzahlen aufnimmt und als Binärzahlen ausgibt:

```
P14.8.3
TSTBIN 3000  CALL INBYTE     CD 57 AF
       3003  LD   C,A        4F           ;Zwi'speich.
       3004  CALL PRINTS     CD 0C 00
       3007  CALL BITBIT     CD 1B 30
       300A  CALL BITBIT     CD 1B 30
       300D  CALL PRINTS     CD 0C 00
       3010  CALL BITBIT     CD 1B 30
       3013  CALL BITBIT     CD 1B 30
       3016  CALL LETNL      CD 06 00
       3019  JR   TSTBIN     18 E5
BITBIT 301B  LD   B,00       06 00
       301D  SLA  C          CB 21
       301F  RL   B          CB 10
       3021  SLA  B          CB 20
       3023  SLA  B          CB 20
       3025  SLA  B          CB 20
       3027  SLA  C          CB 21
       3029  RL   B          CB 10
       302B  LD   A,B        78
       302C  CALL OUTBYT     CD 30 AF
       302F  RET             C9
```

A14.8.4 Überlegen Sie, ohne zum Programmanfang zu schauen:

a) Was enthält der Programmzähler (PC) zu Beginn des relativen Sprungs in 3019?

b) Wie errechnet sich aus PC und dem Offset E5 die Zieladresse dieses Sprungs?

A14.8.5 Schreiben Sie P14.8.3 zur EBS-Routine OUTBIN um, der das auszugebende Byte im Akku übergeben wird. Welches Register muß während des Ablaufs gesichert werden? – Der Raum AEF8...AF1F genügt.

A14.8.6 Schreiben Sie ein Kurzprogramm zum Testen von OUTBIN!

15
Umwandeln von einem Zahlensystem in ein anderes

In der Praxis des Programmierens hat man es vor allem mit Binär-, Dezimal- und Hexadezimalzahlen zu tun. Häufig sind Zahlenwerte aus der einen Darstellung in die andere umzurechnen. Neben den genannten lassen sich beliebig viele andere Zahlensysteme konstruieren.

15.1 Umwandeln einer Hexadezimalzahl in eine Dezimalzahl

Man kann eine Hexadezimalzahl verdoppeln (vervierfachen, verachtfachen usw.), indem man sie um 1 (2, 3 usw.) Stellen nach links shiftet. Der Leser denke sich dazu Beispiele aus und realisiere sie mit geeigneten Programmen.

Man kann eine Dezimalzahl nicht immer durch Shiften (und Rotieren) verdoppeln, weil nicht alle Bits gleichberechtigt sind bzw. weil dabei «Pseudotetraten» auftreten können; das sind Bitmuster, die keiner dezimalen Ziffer entsprechen. Man kann aber Dezimalzahlen verdoppeln, indem man sie zu sich selbst addiert und dann mit DAA dezimal adjustiert. Addiert man mit ADC, dann kommt zur Summe – je nach Carry – noch 1 hinzu. Das läßt sich praktisch anwenden, denn man kann ja den Wert z. B. der Binärzahl 10110 auch so berechnen:

$$
\begin{aligned}
10110 &= 1 * 16 + 0 * 8 + 1 * 4 + 1 * 2 + 0 * 1 \\
&= (((1 * 2 + 0) * 2 + 1) * 2 + 1) * 2 + 0
\end{aligned}
$$

Mit anderen Worten: Man kann eine Binärzahl auch als Polynom auffassen und seinen Wert nach dem Hornerschen Schema berechnen, indem man wiederholt verdoppelt und je nach Maßgabe der Binärzahl 0 oder 1 addiert. Wir benutzen das zu einer trickreichen Umwandlung von hexadezimal (vierstellig) nach dezimal (sechsstellig). Vgl. dazu auch E. Flögel in «Elcomp» 2/83, S. 97.

Unser Programm soll die mit MSB in 1200 und LSB in 1201 gut lesbar angeordnete vierstellige Hexadezimalzahl in eine Dezimalzahl umwandeln und diese – ebenfalls gut lesbar angeordnet – nach 1202...1204 ablegen. Wir setzen es wieder aus einzelnen Abschnitten zusammen:

P15.1.1 shiftet die Hexadezimalzahl um 1 bit nach links und transportiert Bit15 ins Carry:

```
P15.1.1
SHILKS 2100   LD    HL,1201      21 01 12
       2103   SLA   (HL)         CB 26
       2105   DEC   HL           2B
       2106   RL    (HL)         CB 16
       2108   RET                C9
```

P15.1.2 setzt das Zielfeld auf null:

```
P15.1.2
ZFNULL 2120   XOR   A            AF              ;A:=0
       2121   LD    HL,1202      21 02 12
       2124   LD    (HL),A       77
       2125   INC   HL           23
       2126   LD    (HL),A       77
       2127   INC   HL           23
       2128   LD    (HL),A       77
       2129   RET                C9
```

P15.1.3 verdoppelt den Inhalt der Ergebnisbytes durch Dezimaladdition mit Carry:

```
P15.1.3
ZFDOPC 2140   LD    HL,1204      21 04 12
       2143   CALL  SUBRUT       CD 4D 21
       2146   CALL  SUBRUT       CD 4D 21
       2149   CALL  SUBRUT       CD 4D 21
       214C   RET                C9
SUBRUT 214D   LD    A,(HL)       7E
       214E   ADC   A,(HL)       8E
       214F   DAA                27
       2150   LD    (HL),A       77
       2151   DEC   HL           2B
       2152   RET                C9
```

Damit vereinfacht sich das Hauptprogramm zu

```
P15.1.4
HEXDEZ  2000   CALL ZFNULL        CD 20 21
        2003   LD   B,10          06 10       ;16mal
LABEL   2005   CALL SHILKS        CD 00 21
        2008   CALL ZFDOPC        CD 40 21
        200B   DJNZ LABEL         10 F8
        200D   JP   DUMP12        C3 DF AF
```

Wir überzeugen uns von der korrekten Arbeit des Programms. Kontrollbeispiele könnten u. a. sein:

FFFFh = 65535d, 1234h = 4660d. Denken Sie sich weitere aus!

A15.1.5 Machen Sie dazu ein «schönes» Hauptprogramm mit Eingabe durch IN4HEX und Ausgabe durch OUT6HX. Welche Programmteile sind dazu zu ändern?

Für das EBS bauen wir das Programm noch etwas komfortabler aus:

```
P15.1.6
HEXDEZ   AE92   11 E1 AE CD ED AF 21 01 12 CD 4C AF
         AE9E   CD 0C 00 CD B9 AE 06 10 CD C4 AE CD
         AEAA   CD AE 10 F8 21 04 12 CD 20 AF CD 06
         AEB6   00 18 DF
HDNULL   AEB9   AF 06 03 21 02 12 77 23 10 FC C9
HDSHLK   AEC4   21 00 12 CB 26 23 CB 16 C9
HDDOPC   AECD   21 02 12 CD DC AE 23 CD DC AE 23 CD
         AED9   DC AE C9
SRDOPC   AEDC   7E 8E 27 77 C9
TABLE    AEE1   16 55 B3 A3 A1 B0 9C B8 A5 B0 97 20
         AEED   48 45 58 20 2D 3E 20 44 45 5A 0D
```

15.2 Eine andere Hex-Dez-Umwandlung

Wir stellen zunächst eine weniger «schöne» Lösung vor, die auch zum Ziel führt. Der Leser wird dann gebeten werden, das angewendete Verfahren zur Dez-Hex-Umwandlung umzukehren. Auf der Suche nach einem eleganteren Verfahren analysieren wir den Befehl DAA. Zum Abschluß dieses Themas lernen wir eine P15.1.n entsprechende schönere Lösung kennen.

Man kann die Hexadezimalzahl uvwx in eine Dezimalzahl umwandeln, indem man Anfangswert 000000 u-mal 4096, v-mal 256, w-mal 16 und x-mal 1 dezimal hinzuaddiert.

Speicherdisposition:

1200 ... 1201: vierstellige Hexadezimalzahl.
1202 ... 1204: Ergebnisfeld, anfangs alles auf null.
1205 ... 1207: Hier stehen die aktuellen Stellenwerte.

Wieder zunächst die benötigten Unterprogramme:
EFNULL setzt das Ergnisfeld auf Null.
HXSWAP swappt die Hexadezimalzahl um ein Nibble nach links.
SWnnnn: Diese Subroutinen erzeugen den jeweiligen Stellenwert.
ADEFSW addiert zum Ergebnisfeld den momentanen Stellenwert.

```
P15.2.1
EFNULL 2100  LD   HL,1202      21 02 12
       2103  XOR  A            AF
       2104  LD   (HL),A       77
       2105  INC  HL           23
       2106  LD   (HL),A       77
       2107  INC  HL           23
       2108  LD   (HL),A       77
       2109  RET               C9
HXSWAP 210A  XOR  A            AF
       210B  LD   HL,1201      21 01 12
       210E  RLD               ED 6F
       2110  DEC  HL           2B
       2111  RLD               ED 6F
       2113  LD   B,A          47
       2114  RET               C9
SW4096 2115  XOR  A            AF
       2116  LD   (1205),A     32 05 12
       2119  LD   HL,9640      21 40 96
LABLSW 211C  LD   (1206),HL    22 06 12
       211F  RET               C9
SW0256 2120  LD   HL,5602      21 02 56
       2123  JR   LABLSW       18 F7
SW0016 2125  LD   HL,1600      21 00 16
       2128  JR   LABLSW       18 F2
SW0001 212A  LD   HL,0100      21 00 01
       212D  JR   LABLSW       18 ED
ADEFSW 212F  PUSH BC           C5
       2130  LD   B,03         06 03
```

```
        2132 LD   IX,1204     DD 21 04 12
        2136 AND  A           A7
LABADD  2137 LD   A,(IX+00)   DD 7E 00
        213A ADC  A,(IX+03)   DD 8E 03
        213D DAA              27
        213E LD   (IX+00),A   DD 77 00
        2141 DEC  IX          DD 2B
        2143 DJNZ LABADD      10 F2
        2145 POP  BC          C1
        2146 RET              C9
```

Diese Routinen koordiniert das Hauptprogramm:

```
P15.2.2
HXDEZ2 2000 CALL EFNULL     CD 00 21
       2003 CALL HXSWAP     CD 0A 21
       2006 AND  A          A7          ;A=0?
       2007 JR   Z,LABEL2   28 08
       2009 CALL SW4096     CD 15 21
LABEL1 200C CALL ADEFSW     CD 2F 21
       200F DJNZ LABEL1     10 FB
LABEL2 2011 CALL HXSWAP     CD 0A 21
       2014 AND  A          A7          ;A=0?
       2015 JR   Z,LABEL4   28 08
       2017 CALL SW0256     CD 20 21
LABEL3 201A CALL ADEFSW     CD 2F 21
       201D DJNZ LABEL3     10 FB
LABEL4 201F CALL HXSWAP     CD 0A 21
       2022 AND  A          A7          ;A=0?
       2023 JR   Z,LABEL6   28 08
       2025 CALL SW0016     CD 25 21
LABEL5 2028 CALL ADEFSW     CD 2F 21
       202B DJNZ LABEL5     10 FB
LABEL6 202D CALL HXSWAP     CD 0A 21
       2030 AND  A          A7          ;A=0?
       2031 JR   Z,LABEL8   28 08
       2033 CALL SW0001     CD 2A 21
LABEL7 2036 CALL ADEFSW     CD 2F 21
       2039 DJNZ LABEL7     10 FB
LABEL8 203B JP   DUMP12     C3 DF AF
```

Ein Vergleich mit P15.1 stellt P15.2 weit in den Schatten. P15.2 war aber auch nicht als Empfehlung gedacht, sondern lediglich als *Darstellung* einer Variante mit dem Ziel, auch aus dieser zu lernen.

A15.2.3 Schreiben Sie ein Programm, das auf nahezu gleichem Wege von Dezimal nach Hexadezimal umwandelt und dabei die Fähigkeit der Z80-CPU zur Addition vierstelliger Hexadezimalzahlen nutzt. Vereinbarung: Stellenwert in DE und Addition nach IY.

15.3 Überlegungen zum Befehl DAA

36 + 48 = 84; das ist eine Binsenweisheit. Wie aber kommt der Prozessor zu diesem Ergebnis, weil er doch hexadezimal addiert? Er berechnet also 36 + 48 = 7E, und dieses Hexadezimalergebnis wird durch DAA in 84d überführt. Man kann das offensichtlich durch Addition von 6 erzielen:

36 + 48 = 7E	7E + 06 = 84
36 + 48 = 7E	7E + 06 = 84
36 + 49 = 7F	7F + 06 = 85

Diese Sechseraddition muß stattfinden, wenn sich bei der Addition in einem Nibble eine Stellensumme > 9 ergibt. Im Fall FA + 06 = 100 würde aber der Übertrag aus einer 8-bit-Addition im Carry stehen, und dieses benötigen wir für Vergleichsoperationen! Wir weichen daher wie folgt auf eine 16-bit-Addition aus:

1. Wir setzen H gleich null und kopieren die dezimal zu adjustierende Hexadezimalzahl aus dem Akku nach L.
2. Wenn das LSN von L > 9 ist, wird der Inhalt von HL um 0006 erhöht.
3. Der gültige Wert von HL wird auf den Stapel gerettet.
4. HL wird um 4 Binärstellen nach links geshiftet.
5. Feststellung, ob H > 9.
6. Gültigen Wert von HL wiederherstellen.
7. Wenn «5», dann HL um 0060 erhöhen.
8. Zur Ausgabe aus H und aus L.

Für die beiden Vergleiche benutzen wir den Befehl CP n. Dazu müssen Sie noch wissen, daß nach der Subtraktion im Akku (vgl. Abschnitt 5.4) Carry gesetzt ist, wenn sich ein (negativer) Übertrag ergab:

A = 0B	CP 0A → 0B − 0A = 01	CY = 0
A = 0A	CP 0A → 0A − 0A = 00	CY = 0
A = 09	CP 0A → 09 − 0A = FF	CY = 1

Aus diesen Überlegungen ergibt sich schließlich folgende DAA-Simulation:

```
P15.3.1
SUBDAA 4100   LD    H,00        26 00
       4102   LD    L,A         6F
       4103   AND   0F          E6 0F
       4105   CP    0A          FE 0A
       4107   JR    C,LAB1      38 04
       4109   LD    DE,0006     11 06 00
       410C   ADD   HL,DE       19
LAB1   410D   PUSH  HL          E5
       410E   ADD   HL,HL       29
       410F   ADD   HL,HL       29
       4110   ADD   HL,HL       29
       4111   ADD   HL,HL       29
       4112   LD    A,H         7C
       4113   CP    0A          FE 0A
       4115   POP   HL          E1
       4116   RET   C           D8
       4117   LD    DE,0060     11 60 00
       411A   ADD   HL,DE       19
       411B   RET               C9
```

Diese Subroutine ist 28 Bytes lang, der Befehl DAA nur ein Byte, und er bewältigt das Entsprechende in einer Mikrosekunde (= eine millionstel Sekunde)!

Und nun das Kurzprogramm zum Ausprobieren:

```
P15.3.2
TSTDAA 4000   CALL  LETNL       CD 06 00
       4003   CALL  INBYTE      CD 57 AF
       4006   PUSH  AF          F5
       4007   CALL  PRINTS      CD 0C 00
       400A   POP   AF          F1
       400B   CALL  SUBDAA      CD 00 41
       400E   LD    A,H         7C
       400F   CALL  OUTBYT      CD 30 AF
       4012   LD    A,L         7D
       4013   CALL  OUTBYT      CD 30 AF
       4016   JR    TSTDAA      18 E8
```

Probieren Sie TSTDAA mit vielen Beispielen aus und bedenken Sie

dabei, daß die Dezimalzahl xy den Wert x ⋆ 10 + y hat, auch wenn die Nibbles mit Hexadezimalziffern besetzt sind. So ist z. B.

$$FBd = 15 \star 10 + 11 = 161d$$

15.4 Warum es keinen Befehl HAA gibt

Mathematisch gesehen läßt sich DAA als Funktion auffassen, denn jeder auszuwertenden Hexadezimalzahl wird völlig eindeutig genau eine Dezimalzahl zugeordnet. Man könnte daher auch schreiben DAA(FA) = 160 usw. Der höchste Dezimalwert, der so zu erhalten ist, ist DAA(FF) = 165, d. h., der Wertevorrat umfaßt 166 Elemente, der Definitionsbereich hingegen 256. Deshalb müßte eine Umkehrung gegen das Prinzip der Eindeutigkeit verstoßen.

15.5 Verbesserte Dez-Hex-Umwandlung

Aus dem in Abschnitt 15.1 genannten Grund dürfen wir nicht bitweise vorgehen, sondern müssen mit Nibbles operieren. Dabei hilft uns wieder das Hornersche Schema, dieses Mal in der Form

$$6543 = ((6 \star 10 + 5) \star 10 + 4) \star 10 + 3$$

Die Hexadezimaladdition der einzelnen Dezimalziffern ist kein Problem, und die hexadezimale Verzehnfachung ließe sich CPU-gerecht so gestalten:

$$\text{Zahl} \star 10d = (\text{Zahl} \star 2 \star 2 + \text{Zahl}) \star 2$$

Der Befehl ADD HL,HL ist zulässig, und ADD HL,DE kennen wir bereits.

Nach dieser Skizze des Algorithmus geben wir gleich das Programm fertig fürs EBS an. Dieses Mal als (nicht ganz vollständiges) Assemblerprotokoll:

```
P15.5.1
01 0000                LETNL:  EQU  0006
02 0000                PRINTS: EQU  000CH
03 0000                OUT4HX: EQU  AF25H
04 0000                IN6HEX: EQU  AF46H
05 0000                MSGNL:  EQU  AFEDH
06 0000                        REL  AE34H
07 AE34                ;
08 AE34 11 7B AE       DEZHEX: LD   DE,TABLE
09 AE37 CD ED AF               CALL MSGNL
10 AE3A 21 02 12       LABEL0: LD   HL,1202H
11 AE3D CD 47 AF               CALL IN6HEX
12 AE40 CD 0C 00               CALL PRINTS
13 AE43 CD 6C AE               CALL DHSWLK
14 AE46 21 00 00               LD   HL,0000
15 AE49 06 05                  LD   B,05
16 AE4B CD 65 AE       LABEL1: CALL HLZEHN
17 AE4E CD 6C AE               CALL DHSWLK
18 AE51 5F                     LD   E,A
19 AE52 16 00                  LD   D,00
20 AE54 19                     ADD  HL,DE
21 AE55 10 F4                  DJNZ LABEL1
22 AE57 22 00 12               LD   (1200H),HL
23 AE5A 21 01 12               LD   HL,1201H
24 AE5D CD 25 AF               CALL OUT4HX
25 AE60 CD 06 00               CALL LETNL
26 AE63 18 D5                  JR   LABEL0
27 AE65 E5             HLZEHN: PUSH HL
28 AE66 29                     ADD  HL,HL
29 AE67 29                     ADD  HL,HL
30 AE68 D1                     POP  DE
31 AE69 19                     ADD  HL,DE
32 AE6A 29                     ADD  HL,HL
33 AE6B C9                     RET
34 AE6C E5             DHSWLK: PUSH HL
35 AE6D AF                     XOR  A
36 AE6E 21 00 12               LD   HL,1200H
37 AE71 ED 6F                  RLD
38 AE73 23                     INC  HL
39 AE74 ED 6F                  RLD
40 AE76 23                     INC  HL
41 AE77 ED 6F                  RLD
42 AE79 E1                     POP  HL
43 AE7A C9                     RET
44 AE7B 16             TABLE:  DEFB 16H
45 AE7C 55 B3 A3 A1            DEFM 'Umwandlung'
46 AE80 B0 9C B8 A5
```

```
47 AE84 B0 97
48 AE86 20 44 45 5A           DEFM ' DEZ -> HEX'
49 AE8A 20 2D 3E 20
50 AE8E 48 45 58
51 AE91 0D                     DEFB 0DH
52 AE92                ;
53 AE92                        END
```

Sie haben kein Lehrbuch der Assemblersprache vor sich; dennoch sollen Sie einige Einblicke gewinnen. Die folgenden Fragen sind nur als Verständnis- und Lesehilfe gedacht:

A15.5.2

a) Was bedeutet das z. B. in den Zeilen 02 bis 06 den Zahlen angehängte H?
b) Weshalb durfte es z. B. in den Zeilen 01, 14, 15 weggelassen werden?
c) Welcher Assemblerbefehl fügt ohne weitere Verarbeitung ein Byte in das Programm ein?
d) Welcher Assemblerbefehl wandelt ausgeschriebene Wörter in ASCII-Strings um?

P15.5.1 wurde nur für Hexadezimalzahlen bis FFFF ausgelegt!

16
Diverse Einzelthemen

16.1 Das Ermitteln der Prüfsumme

Bei der Eingabe längerer Programme kann man sich leicht vertun, aber man findet den Fehler nicht so leicht wie bei einem BASIC-Programm, denn der Objektcode (das aus Hexadezimalzahlen bestehende Maschinenprogramm) ist nicht so leicht zu überprüfen wie ein sinnerfüllter BASIC-Quelltext. Man gibt deshalb für längere Programme manchmal Prüfsummen an. Eine Prüfsumme ist die Summe aller Hexadezimalzahlen, die ein Maschinenprogramm bilden. Ein absoluter Schutz ist das nicht, denn wir könnten ja einmal um eins zuviel und einmal um eins zuwenig eingetippt haben. Doch die Wahrscheinlichkeit dafür, daß sich zwei Fehler gegenseitig aufheben, ist recht gering. Die größte Prüfsumme, die wir in unserem MZ-700 erreichen könnten, läge vor, wenn alle 10000h Bytes mit dem Höchstwert FFh belegt wären. 10000h⋆FF=FF0000. Wir kommen also in jedem Fall aus, wenn wir die Prüfsumme in drei Bytes berechnen.

Wir statten auch dieses Programm mit einigem Komfort aus, greifen aber bis auf weiteres auf die gewohnte Darstellungsweise zurück.

Speicherdisposition:
1200 (+1): Hierin wird mittels IN4HEX die VON-Adresse geladen, von der an die Prüfsumme gebildet werden soll. Mit jedem aufaddierten Byte wird der Inhalt dieser Speicherstelle(n) um eins erhöht.
1202 (+1): Hier steht die BIS-Adresse.
1204 ... 1206: Hier wird die Prüfsumme gebildet.

Zuerst stellen wir die Dialogtexte bereit:

```
P16.1.1
TABPRS  2800  20 50 9D AD AA A4 A5 B3 B3 92 20 0D
TABVON  280C  AB B7 B0 20 0D
TABBIS  2811  20 9A A6 A4 20 0D
```

Dann organisieren wir den Eingabedialog und die Eingaben und setzen die Prüfsumme auf null:

```
P16.1.2
PREINL 2100   CALL LETNL          CD 06 00
       2103   LD   DE,TABVON      11 0C 28
       2106   CALL MSG            CD 15 00
       2109   LD   HL,1201        21 01 12
       210C   CALL IN4HEX         CD 4C AF
       210F   LD   DE,TABBIS      11 11 28
       2112   CALL MSG            CD 15 00
       2115   LD   HL,1203        21 03 12
       2118   CALL IN4HEX         CD 4C AF
       211B   XOR  A              AF
       211C   LD   (1204),A       32 04 12
       211F   LD   (1205),A       32 05 12
       2122   LD   (1206),A       32 06 12
       2125   RET                 C9
```

Der üppige Befehlsreichtum der Z80-CPU läßt für viele (Teil-)Probleme mehrere Lösungen zu. Wenn man in ein EPROM möglichst viel Programm hineinbrennen will, wird man die kürzeste Lösung suchen – bei zeitkritischen Problemen die schnellste. Wir legen mehr Wert auf Lesbarkeit und zeilenmäßig kürzere Versionen.

Wir addieren nun den Inhalt der jeweils adressierten Speicherstelle ohne Carry zu (1204). Die Überträge nach 1205 und 1206 werden durch Addieren von null mit Carry erzielt:

```
P16.1.3
ADDIRN 2140   LD   A,(1204)       3A 04 12
       2143   ADD  A,(HL)         86
       2144   LD   (1204),A       32 04 12
       2147   LD   A,(1205)       3A 05 12
       214A   ADC  A,00           CE 00
       214C   LD   (1205),A       32 05 12
       214F   LD   A,(1206)       3A 06 12
       2152   ADC  A,00           CE 00
       2154   LD   (1206),A       32 06 12
       2157   RET                 C9
```

Das Hauptprogramm gibt zunächst den Namen der Utility aus und ruft dann PREINL und ADDIRN auf. HL wird danach auf die nächste Speicherstelle gerichtet. Nun wird festgestellt, ob deren Inhalt auch

noch zu addieren ist oder ob die BIS-Adresse überschritten wurde. Selbstverständlich wird man das normalerweise mit einer 16-bit-Subtraktion ermitteln. Hier soll jedoch beiläufig gezeigt werden, welche Schritte ohne diese zu vollziehen wären: Wurde H größer als der Inhalt von 1203, ist nach 2016 Carry gesetzt, und die Additionsschleife wird verlassen. Im Fall H = (1203) (nur der kommt als Alternative in Frage) ist L mit (1202) zu vergleichen. Wenn L 〉 (1202), dann ist nach 201E Carry gesetzt, und es geht zur Ausgabe.

```
P16.1.4
PRUESU 2000  CALL CLS          CD 9F AF
       2003  LD   DE,TABPRS    11 00 28
       2006  CALL MSGNL        CD ED AF
LPRUS0 2009  CALL PREINL       CD 00 21
       200C  LD   HL,(1200)    2A 00 12
LPRUS1 200F  CALL ADDIRN       CD 40 21
       2012  INC  HL           23
       2013  LD   A,(1203)     3A 03 12
       2016  CP   H            BC
       2017  JR   C,LPRUS2     38 08
       2019  JR   NZ,LPRUS1    20 F4
       201B  LD   A,(1202)     3A 02 12
       201E  CP   L            BD
       201F  JR   NC,LPRUS1    30 EE
LPRUS2 2021  LD   DE,TABPRS    11 00 28
       2024  CALL MSG          CD 15 00
       2027  LD   HL,1206      21 06 12
       202A  CALL OUT6HX       CD 20 AF
       202D  JR   LPRUS0       18 DA
```

A16.1.5 Testen Sie das Programm gewissenhaft aus. Ermitteln Sie z. B. die Prüfsummen von ... bis

a) 0000...0000 (kein Irrtum!),
b) 0000...0100 und 0000...0101.
c) Berechnen Sie aus b1) und b2) den Inhalt von 0101.
d) 0000...0800, 0800...0FFF, 0000...0FFF (Monitor).
e) Berechnen Sie aus den Ergebnissen von d) den Inhalt von 0800. Prüfen Sie mit Dxxxx oder Mxxxx nach.
f) Fügen Sie dieses nützliche Programm ins EBS ein.

16.2 Blocktransfer

Sie haben eben manuell vier Programmblöcke im Speicher verschoben und mit dem EBS zusammengefügt. Die dabei im Innern dieser Blöcke zu verändernden Adressen haben wir von Hand korrigiert und werden das auch weiterhin tun. Das bloße Verschieben soll jetzt jedoch durch ein etwas aufwendigeres Programm erledigt werden.

A16.2.1 Entwerfen Sie unter ausschließlicher Verwendung der bisher benutzten Befehle Programme, die folgendes leisten:

a) DUMP12 (AFDF...AFEC) soll nach 1200... kopiert werden. Empfehlung: Rufen Sie vorher NULL12 mit JAFF4.
b) Der Block 1200...1204 soll nach 1203...1207 verschoben werden. Anfangszustand: 1200 AA BB CC DD EE 11 11 11.
c) Der Block 1203...1207 soll nach 1200...1204 verschoben werden. Anfangszustand: 1200 11 11 11 AA BB CC DD EE.

Arbeiten Ihre Programme, insbesondere zu 1. und 2. wirklich einwandfrei? Sind alle Einsen verschwunden?

Beim Blocktransfer sind drei Fälle zu unterscheiden:

```
            Fall 1          Fall 2          Fall 3

         +--------+       +--------+      +---------+
AAA0     ! ANFalt!        !ANFneu  !      ! ANFalt  !
         !+--------++    ++--------+!      !         !
BBB0     !!ANFneu!!      !!ANFalt!!       ! ENDalt  !
         !!      !!      !!      !!       +---------+
         !!      !!      !!      !!       +---------+
CCC0     !!ENDalt!!      !!ENDneu!!       ! ANFneu  !
         ++--------+!     !+--------++     !         !
DDD0      !ENDneu  !      ! ENDalt!       ! ENDneu  !
          +--------+      +--------+      +---------+
```

Bild 16.1 Die drei Fälle beim Blocktransfer

Das Verschieben von Speicherblöcken kommt in der Praxis des Programmierens in Maschinensprache nicht gerade selten vor: Manchmal will man eine fertige Routine in ein aufzubauendes Programm übernehmen, manchmal will man ein unveränderliches Programm aus dem ROM ins RAM schieben (vgl. Abschnitt 24.5), um es dort zu verändern. Der Z80 stellt hierfür zwei besonders komfortable Befehle bereit:

LDIR überträgt zuerst das erste Byte eines Blocks, dann das zweite, das dritte usw.
LDDR überträgt zuerst das letzte, das vorletzte usw.
Dazu muß
die jeweils erste Her-Adresse in HL,
die jeweils erste Hin-Adresse in DE und
die Anzahl der zu kopierenden Bytes in BC
eingeschrieben sein. Man kann so bis zu 64 KB mit einem Vierzeiler transferieren!

Wir lösen jetzt A16.2.1b auf diesem Wege:

```
P16.2.2
4A00  LD    HL,1204             21 04 12
4A03  LD    DE,1207             11 07 12
4A06  LD    BC,0005             01 05 00
4A09  LDDR                      ED B8
4A0B  JP    DUMP12              C3 DF AF
```

A16.2.3 Lösen Sie A16.2.1c) entsprechend!

Wie schon angedeutet, bewirken die Blocktransferbefehle lediglich die bloße Verschiebung. Enthält der verschobene Block auch noch absolute Adressen – z. B. bei JP, CALL, LD HL,nn usw. –, so ist zu prüfen, ob diese auch verändert werden müssen. Wir tun das von Hand. Es gibt aber auch Relokatierprogramme, die das bewerkstelligen. Sie sind zu kompliziert, als daß wir sie in diesem Buch besprechen könnten.

16.3 Ein Blick in den «richtigen» Stapel (Systemstapel)

In Kapitel 10 haben wir uns schon mit dem Stapel beschäftigt. Ein Blick in den vom Monitor benutzten «richtigen» Stapel war bisher kaum möglich, da er ja nur mit Hilfe des Monitors möglich wäre, und der würde ihn ja beim bisherigen Vorgehen neu einrichten. Jetzt kopieren wir den Stapel vorher in einen Bereich, der nach RST 00 nicht überschrieben wird. P16.3.1 dient nur dazu, einen überschaubaren Zustand des Stapels herbeizuführen und dann alles weitere zu stoppen.

```
P16.3.1  (Prüfsumme: 0CBA)
3000  CALL 3008                    CD 08 30
3003  NOP NOP NOP NOP NOP          00 00 00 00 00
3008  CALL 3010                    CD 10 30
300B  NOP NOP NOP NOP NOP          00 00 00 00 00
3010  CALL 3018                    CD 18 30
3013  NOP NOP NOP NOP NOP          00 00 00 00 00
3018  CALL 3020                    CD 20 30
301B  NOP NOP NOP NOP NOP          00 00 00 00 00
3020  LD    DE,1234                11 34 12
3023  PUSH DE                      D5
3024  LD    DE,5678                11 78 56
3027  PUSH DE                      D5
3028  LD    DE,9ABC                11 BC 9A
302B  PUSH DE                      D5
302C  LD    HL,1000                21 00 10
302F  LD    DE,1200                11 00 12
3032  LD    BC,00F0                01 F0 00
3035  LDIR                         ED B0
3037  HALT                         76
```

A16.3.2 Bevor Sie das Programm eingeben und laufen lassen, überlegen Sie doch bitte folgende Fragen:

a) Von wo bis wo reicht der Stack?
b) In welchen Bereich wird der Stapel kopiert?
c) Wo erwarten Sie den ersten Eintrag? Sehen Sie bitte vor Beantwortung dieser Frage genau ins Z80-Handbuch!
d) Können Sie vielleicht den Inhalt der Stack-Kopie voraussagen?

16.4 Zufallszahlen aus eigenen Algorithmen

Zufallszahlen spielen in der Computerei eine nicht zu unterschätzende Rolle. Sie werden für vielfältige statistische Zwecke (Stichproben, aber auch Ermitteln der Kreiszahl Pi nach dem Monte-Carlo-Verfahren, Optimierungsprobleme usw.) benötigt; aber auch für Spiele, bei denen es auf Überraschungseffekte ankommt.

Nun ist ein Computer aber eine voll determinierte Maschine, in der es keinen Zufall gibt. Man kann mit ihm aber sehr wohl Pseudozufallszahlen (falsche oder besser: nicht echte) erzeugen. Alles, was wir benötigen, ist ein Anfangswert und eine Rechenvorschrift, nach der aus dem momentanen Wert der folgende berechnet wird.

Eine derartige Rechenvorschrift könnte im allereinfachsten Falle so aussehen: Der neue Zufallswert (Randomzahl) wird aus der alten durch Verdoppeln erzeugt. Verdoppeln einer Hexadezimalzahl und Linksshiften sind, wie wir wissen, weithin identisch. Wenn wir von einer 2-Byte-Zahl ausgehen, können wir leicht vorhersagen, daß sie nach höchstens 16maligem Verdoppeln bei Mißachtung des Übertrags zu null wird. Damit das nicht geschieht, addieren wir mit Carry.

Bei der Realisierung verwenden wir einen neuen Befehl, und um den zu verstehen, müssen wir noch einen Blick auf die Architektur des Z80 werfen:

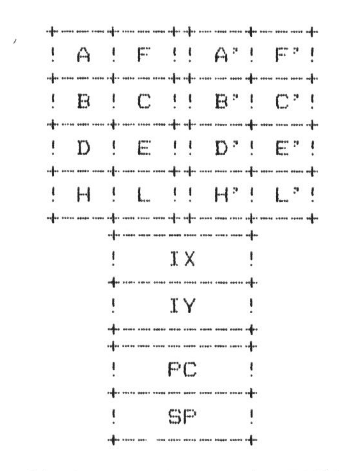


Bild 16.2 Die Register des Z80(A)

Bild 16.2 zeigt, daß vier Registerpaare des Z80 doppelt vorhanden sind; man kann die nebeneinanderstehenden Paare gegeneinander vertauschen, z. B. durch den Befehl EX AF,AF'. Wir wenden diesen Austausch (exchange) im folgenden Programm an, um (zusammen mit den beiden A-Registern (Akkus)) in F (bzw. F') den bei der Addition erhaltenen Carry-Zustand für die nächste Addition aufzubewahren. Damit erhält unser Programm folgende Gestalt:

```
P16.4.1   (Prüfsumme: 1087)

RANDOM 2000  CALL INBYTE     CD 57 AF
       2003  LD   H,A        67
       2004  CALL INBYTE     CD 57 AF
       2007  LD   L,A        6F
LABEL  2008  CALL LETNL      CD 06 00
       200B  LD   A,H        7C
       200C  CALL OUTBYT     CD 30 AF
       200F  LD   A,L        7D
       2010  CALL OUTBYT     CD 30 AF
       2013  EX   AF,AF'     08
       2014  ADC  HL,HL      ED 6A
       2016  EX   AF,AF'     08
       2017  CALL PAUSWK     CD D4 AF
       201A  CALL SBMONI     CD B3 AF
       201D  JR   LABEL      18 E9
```

Nach dem Programmstart mit J2000 wird der vierstellige Anfangswert eingegeben. Nach jedem Durchlauf muß eine Taste gedrückt werden, um die nächste Zufallszahl zu erhalten. Drückt man SHIFT und BREAK gleichzeitig, so bringt uns SBMONI in den Monitor zurück.

Je nach zufälligem Carry-Anfangszustand erhalten wir mit den Anfangswerten 0001, 0002, 0500, 3412 zyklische Folgen mit der Periodenlänge 17. Muß das immer so sein? (Wir haben de facto ein 17-Bit-Schieberegister aufgebaut!)

A16.4.2 Versuchen Sie, ob man beim Verdreifachen (z. B. mittels DE) längere Perioden erhält!

Weil wir schon bald einen eleganteren Weg zum Erzeugen von Zufallszahlen kennenlernen werden, brauchen wir diese simplen Erstansätze nicht weiter zu verfolgen.

16.5 Die eingebaute Uhr

Der MZ-700 ist mit einem Zeitgeberbaustein 8253 ausgestattet, dessen Signale u. a. als Uhr von hoher Ganggenauigkeit aufgefaßt werden können. Sie wird mit Hilfe der Monitorroutine TIMST (0033) gestellt und gestartet und mittels TIMRD (003B) gelesen (vgl. Sharp-Handbuch Seite 153). Ein Tag hat 24 ⋆ 60 ⋆ 60 = 86 400 Sekunden.

Zur Darstellung dieses Betrages als Hexadezimalzahl benötigt man fünf Stellen: 86400d = 15180h, deren erste nur 0 oder 1 sein kann. Im MZ-700 zeigt der Akku an, ob es sich um eine Vormittagszeit (A = 0) oder um eine Nachmittagszeit (A = 1) handelt. Die übrigen 4 Hexadezimalstellen nimmt das DE-Register auf.

Angenommen, wir wollen um 19h 23min 45s die Uhr stellen. Dann wäre «Nachmittag» als A = 1 zu codieren und übrig bliebe 7h 23min 45s.

Wegen 7 * 60 * 60 + 23 * 60 + 45 = 26625d = 6801h ist DE mit 6801 zu laden. Danach muß TIMST aufgerufen werden. Das Stellen einer Uhr hat keinen Sinn, wenn man sie nicht wieder ablesen kann. Wir leiten daher das Programm in eine endlose Schleife, in der ständig durch TIMRD (003B) ausgelesen und mittels OUTBYT angezeigt wird. Die Schleife kann durch SBMONI unterbrochen werden.

```
P16.5.1   (Prüfsumme: 0CBE)

UHROUT 2000  LD    A,01        3E 01
       2002  LD    DE,6801     11 01 68
       2005  CALL  TIMST       CD 33 00
LOOP   2008  CALL  LETNL       CD 06 00
       200B  CALL  TIMRD       CD 3B 00
       200E  CALL  OUTBYT      CD 30 AF
       2011  LD    A,D         7A
       2012  CALL  OUTBYT      CD 30 AF
       2015  LD    A,E         7B
       2016  CALL  OUTBYT      CD 30 AF
       2019  CALL  SBMONI      CD B3 AF
       201C  JR    LOOP        18 EA
```

Das Programm läuft ziemlich rasch ab; je Sekunde werden auf dem Bildschirm etwa 20 Zeilen ausgegeben, d. h., die Schleife (2008 ... 201D) wird in etwa 50 000 μs durchlaufen. Wenn man bedenkt, daß bei einer Taktfrequenz von 3,5 MHz in sehr grober Annäherung je Mikrosekunde rund ein Byte des Maschinenprogramms abgearbeitet wird, so erscheint der Programmdurchlauf langsam. Man muß jedoch dabei bedenken, daß sich hinter fast allen aufgerufenen Routinen längere Programmteile verbergen, die z. T. mehrmals durchlaufen werden.

Unser erstes, noch recht primitives Uhrprogramm läuft korrekt. Leider aber kann man mit dieser Form der Zeitansage wenig anfangen. Versuchen wir also, die Anzeige lesbarer zu gestalten!

Formatierte Zeitanzeige

Um aus der hexadezimal angezeigten Anzahl der Sekunden auf Stunden umzurechnen, müssen wir durch 3600 dividieren. Höchste Zeit also, die vierte Grundrechenart nachzutragen. In Maschinenprogrammen dividiert man im allgemeinen, indem man den Teiler (Divisor) von der zu teilenden Zahl (Dividend) so oft subtrahiert, bis ein negativer Übertrag (CY = 1) erfolgt, also einmal zu oft. Dann wird einmal zurückaddiert und bei alldem natürlich die Anzahl der Subtraktionen mitgezählt, denn sie ergibt ja den gesuchten Quotienten, den wir in diesem Fall als Dezimalzahl benötigen.

Wir entwerfen die Division gleich passend für das Uhrzeitproblem. Am Schluß soll in 1200 die Anzahl der Stunden stehen, in 1201 die Anzahl der Minuten und in 1202 die der Sekunden. Damit ergibt sich folgender Aufbau:

1. 1200 ... 1202 auf null setzen.
2. Uhrzeit lesen.
3. Wenn A = 1, dann (1200) = 12 für Nachmittag.
4. DE nach HL wegen SBC HL,DE.

5a DE = 0E10 = Anzahl Sekunden in einer Stunde.
6a HL = HL − DE und (1200) inkrementieren.
7a Solange CY = 0 zurück nach 6a.
8a HL = HL + DE und (1200) dekrementieren.
5b DE = 003C = Anzahl Sekunden in einer Minute.
6b HL = HL − DE und (1201) inkrementieren.
7b Solange CY = 0 zurück nach 6b.
8b HL = HL + DE und (1201) dekrementieren.
5c DE = 0001. Division durch 1 wegen DAA!
6c HL = HL − DE und (1202) inkrementieren.
7c Solange CY = 0 zurück nach 6c.
8c HL = HL + DE und (1202) dekrementieren.

9. Neue Zeile und Cursor einmal nach oben.
10. Zeit ausgeben.
11. Zurück nach 1.

P16.5.2 (Prüfsumme: 27B7)

```
UHRANZ  2000 LD   A,01        3E 01
        2002 LD   DE,6801     11 01 68
        2005 CALL TIMST       CD 33 00
        2008 CALL CLS         CD 9F AF
NEU     200B CALL NULL12      CD F4 AF     ;1.
        200E CALL TIMRD       CD 3B 00     ;2.
        2011 AND  A           A7           ;Z-Flag?
        2012 JR   Z,L1        28 05
        2014 LD   A,12        3E 12        ;3.
        2016 LD   (1200),A    32 00 12
L1      2019 LD   H,D         62           ;4.
        201A LD   L,E         6B
        201B LD   IX,1200     DD 21 00 12 ;Pointer
        201F LD   DE,0E10     11 10 0E     ;5a
        2022 CALL DIV         CD 47 20
        2025 LD   DE,003C     11 3C 00     ;5b
        2028 CALL DIV         CD 47 20
        202B LD   DE,0001     11 01 00     ;5c
        202E CALL DIV         CD 47 20
        2031 CALL LETNL       CD 06 00     ;9.
        2034 LD   A,12        3E 12
        2036 CALL PRNT        CD 12 00
        2039 LD   HL,1200     21 00 12     ;10.
        203C CALL AUSGEB      CD 64 20
        203F CALL AUSGEB      CD 64 20
        2042 CALL AUSGEB      CD 64 20
        2045 JR   NEU         18 C4        ;11.
DIV     2047 LD   A,(IX+00)   DD 7E 00     ;6abc
        204A ADD  A,01        C6 01
        204C DAA              27
        204D LD   (IX+00),A   DD 77 00
        2050 AND  A           A7           ;CY:=0
        2051 SBC  HL,DE       ED 52
        2053 JR   NC,DIV      30 F2        ;7abc
        2055 LD   A,(IX+00)   DD 7E 00     ;8abc
        2058 SUB  01          D6 01
        205A DAA              27
        205B LD   (IX+00),A   DD 77 00
        205E AND  A           A7           ;CY:=0
        205F ADC  HL,DE       ED 5A
        2061 INC  IX          DD 23
        2063 RET              C9
AUSGEB  2064 LD   A,(HL)      7E
        2065 CALL OUTBYT      CD 30 AF
        2068 CALL PRINTS      CD 0C 00
        206B INC  HL          23
        206C RET              C9
```

Das Programm läuft einwandfrei. Einziger Schwachpunkt ist das unschöne Stellen der Uhr. Es sein dem Leser überlassen, hier für mehr Komfort zu sorgen.

16.6 Zufallszahlen aus dem Intervall-Timer

Die Uhr wird mit Befehl CALL 003B gelesen. Im Monitorlisting finden wir folgende Eintragung:

```
50   003B   C35803      JP   ?TMRD       ;TIME READ
```

Wir haben schon des öfteren Programme nach Assemblerart mit Kommentaren versehen; sie werden vom eigentlichen Programm durch Semikolon abgetrennt, vergleichbar mit den REMs in BASIC-Programmen. Wir haben auch schon symbolische Adressen benutzt, die bei Benutzung eines Assemblers (Hilfsprogramm für die Übersetzung aus der Assemblersprache in die echte Maschinensprache) einen sehr hohen Programmierkomfort ermöglichen. Der Assembler rechnet mit diesen Labels wie mit Variablen, auch wenn ihr endgültiger Wert noch nicht feststeht. Das in 50 003B benutzte Label «?TMRD» finden wir im Sharp-Handbuch auf Seite 201 unter Page 64 in der 13. Zeile als zweiten Eintrag und daneben die zugehörige Adresse 0358. Wir hätten jedoch gar nicht so weit zu blättern brauchen, denn die oben zitierte Zeile besagt in unserer gewohnten Form lediglich

```
003B      JP   0358      C3 58 03
```

Bitte vergleichen Sie genau! Dort steht also ein uns gut bekannter Sprungbefehl, und wir suchen bei 0358 weiter.

Auf Seite 175 finden wir unter Page 12 in Zeile 15 die als Kommentar hervorgehobene Überschrift «TIME READ». In Zeile 22 beginnt der eigentliche Programmabschnitt. Wir stoßen danach mit etwas Spürsinn auf die Zeile 36:

```
036D      JR      C,037F      38 10
```

In diesem Bereich stehen die Befehle DI und EI, zu denen Sie im Anhang A einen kurzen Hinweis finden.

In Verbindung mit Zeile 37 sind dann die Zeilen 47 und 48 besonders aufschlußreich:

```
0380   LD   HL,E006   21 06 E0
0333   LD   A,(HL)    7E
```

Hat die Speicherstelle E006 etwas mit der Uhr zu tun? Auf Seite 130 des Sharp-Handbuchs werden die Speicherstellen E004 ... E007 dem «Zeitgeber 8253» zugeordnet. Ohne auf seine Funktion näher einzugehen, versuchen wir einen Einblick in seine Register.

Wir verfassen nun ein Programm, das die zugehörigen Speicherstellen ausliest und ihren Inhalt auf dem Bildschirm anzeigt. Damit wir folgen können, bauen wir eine Verzögerung ein. Wir sollten nicht vergessen, die Uhr vorsichtshalber neu zu starten!

```
P16.6.1   (Prüfsumme: 13E4)

RD8253  3000  LD    A,01       3E 01      ;Uhr stellen
        3002  LD    DE,6801    11 01 68
        3005  CALL  TIMST      CD 33 00
LOOP    3008  LD    A,(E004)   3A 04 E0
        300B  CALL  OUTBYT     CD 30 AF
        300E  CALL  PRINTS     CD 0C 00
        3011  LD    A,(E005)   3A 05 E0
        3014  CALL  OUTBYT     CD 30 AF
        3017  CALL  PRINTS     CD 0C 00
        301A  LD    A,(E006)   3A 06 E0
        301D  CALL  OUTBYT     CD 30 AF
        3020  CALL  PRINTS     CD 0C 00
        3023  LD    A,(E007)   3A 07 E0
        3026  CALL  OUTBYT     CD 30 AF
        3029  LD    A,40       3E 40      ;variieren
        302B  CALL  WARTEN     CD B8 AF
        302E  CALL  LETNL      CD 06 00
        3031  JR    LOOP       18 D5
```

Wir sehen auf dem Bildschirm in schneller Folge Zahlenkolonnen nach oben wandern. Einige Kolonnen scheinen lediglich sich abwechselnde Werte zu zeigen; doch in der zweiten Kolonne treten Zahlen auf, die keine auffallende Gesetzmäßigkeit erkennen lassen. Für viele Anwendungen werden wir also wohl die Inhalte von E005 als Zufallszahlen lesen können. Vorsichtshalber sehen wir uns E005 vorher noch einmal genauer an:

```
P16.6.2
RDE005 3100 LD   A,01          3E 01          ;Uhr st.
       3102 LD   DE,6801       11 01 68
       3105 CALL TIMST         CD 33 00
       3108 LD   B,A0          06 A0          ;160mal
       310A LD   IX,1200       DD 21 00 12
LOOP   310E LD   A,(E005)      3A 05 E0
       3111 LD   (IX+00),A     DD 77 00
       3114 INC  IX            DD 23
       3116 DJNZ LOOP          10 F6
       3118 JP   DUMP12        C3 DF AF
```

Unsere Vorsicht hat sich gelohnt. Wenn wir E005 in dieser Weise auslesen, erhalten wir etwa folgendes Bild:

1200 FB 3C FB 3C FA 3C FA 3C
1208 F9 3C F9 3C F8 3C F8 3C
1210 F8 3C F7 3C F7 3C F6 3C

Daß sich die Glieder der fallenden Folge FB, FA, F9, F8, ... wiederholen, braucht uns nicht zu beunruhigen, denn unser schnelles Programm ließ dem Intervallgenerator ja gar keine Zeit, rechtzeitig weiterzuticken. Höchst bedenklich ist dagegen die penetrante alternierende Wiederholung des Wertes 3C. Probieren geht nach einem Sprichwort über Studieren. Wandeln wir doch P16.6.2 ein wenig ab:

```
P16.6.3
       3100  .... ........    .. .. ..
LOOP   3108  LD   A,(E005)    3A 05 E0
       310B  CALL OUTBYT      CD 30 AF
       310E  CALL PRINTS      CD 0C 00
       3111  CALL PRINTS      CD 0C 00
       3114  LD   A,20        3E 20
       3116  CALL WARTEN      CD B8 AF
       3119  JR   LOOP        18 ED
```

Bei diesem gemächlichen Tempo ist die störende Erscheinung verschwunden. Dieses Pseudo-Random-Verfahren erscheint brauchbar. Man sollte es aber nicht ohne weitere Überprüfung (vgl. P16.7.1) für kritische statistische Probleme benutzen.

A16.6.4 Der Computer knobelt
Schreiben Sie ein Programm, das einen Würfel simuliert! Der jeweils nächste Wurf soll auf Tastendruck ausgegeben werden. Der Ablauf ist so zu verzögern, daß jedesmal nur eine Ausgabe erfolgt.

16.7 Von der Güte unseres Random-Generators

Vielleicht werden auch in Ihrer Lösung nach einiger Zeit die Zweien knapp. Dies ist Grund genug, die Güte unseres Random-Generators zu prüfen, indem wir z. B. jede 1 in 1201, jede 2 in 1202, ... jede 6 in 1206 durch Inkrementieren zählen. Der jeweilige Zählerstand soll wie bei der Uhrzeit angezeigt werden. Unser Programm weist eine Besonderheit auf: Es verändert sich selbst; das Offset zu IX wird je nach geworfener Augenzahl nach 412C eingespeichert. Nach je hundert Würfen soll der Ablauf unterbrochen und erst auf Tastendruck wieder freigegeben werden. Dadurch soll unkontrolliertes Überzählen verhindert werden.

```
P16.7.1   (Prüfsumme: 1C84 (bei xx = 00))

GUERND 4100  LD    A,01          3E 01
       4102  LD    DE,6801       11 01 68
       4105  CALL  TIMST         CD 33 00
       4108  CALL  NULL12        CD F4 AF
       410B  CALL  CLS           CD 9F AF
LOOP1  410E  LD    B,64          06 64
LOOP2  4110  LD    IX,1200       DD 21 00 12
       4114  LD    A,(E005)      3A 05 E0
       4117  LD    H,00          26 00
       4119  LD    L,A           6F
       411A  LD    DE,0006       11 06 00
       411D  AND   A             A7
L1     411E  SBC   HL,DE         ED 52
       4120  JR    NC,L1         30 FC
       4122  CCF                 3F
       4123  ADC   HL,DE         ED 5A
       4125  LD    A,L           7D
       4126  INC   A             3C
       4127  LD    (L2+02),A     32 2C 41
L2     412A  INC   (IX+xx)       DD 34 xx
       412D  LD    HL,1201       21 01 12
       4130  PUSH  BC            C5
       4131  LD    B,06          06 06
```

```
LOOP3  4133  LD    A,(HL)     7E
       4134  CALL  OUTBYT     CD 30 AF
       4137  CALL  PRINTS     CD 0C 00
       413A  INC   HL         23
       413B  DJNZ  LOOP3      10 F6
       413D  POP   BC         C1
       413E  CALL  LETNL      CD 06 00
       4141  LD    A,12       3E 12      ;Cursor hoch
       4143  CALL  PRNT       CD 12 00
       4146  DJNZ  LOOP2      10 C8
       4148  CALL  PAUSKY     CD D9 AF
       414B  JR    LOOP1      18 C1
```

A16.7.2 Fragen:

a) Was bedeutet L2?
b) Was bedeutet L2+02?
c) Was geben Sie an Stelle von xx ein?
d) Weshalb wird in 4130 das BC-Register gepushet?
e) Von wo bis wo reicht der Programmteil, der die Ausgabe erledigt?
f) Nach dem Starten des Programms vergehen etwa 2,5 s bis zu seinem ersten Zwischenhalt. Wie viele Zufallszahlen werden in dieser Zeit berechnet?
g) Wird nur der jeweils veränderte Zählerstand angezeigt oder alle sechs?
h) Sie könnten die 2-Bytes-Division durch eine 1-Byte-Division ersetzen.

16.8 Vom Refresh-Register

Die Z80-CPU ist in der Lage, dynamische RAMs zu verwalten. Das sind im Prinzip Speicherbausteine, die mikroskopisch kleine Kondensatoren enthalten. Diese sind entweder geladen oder ungeladen. Weil Kondensatoren, insbesondere kleine, ihre Ladung allmählich verlieren, muß diese beim dynamischen RAM in schneller Folge aufgefrischt werden – falls eine zu erhaltende Ladung vorhanden ist. Diesen Vorgang steuert das Refresh-Register, in dem ständig gezählt wird. Man kann es mit dem Befehl *LD A,R* auslesen und den erhaltenen Wert – mit Vorbehalt – als Zufallszahl verwenden. Wir verschaffen uns einen Überblick:

```
P16.8.1
REFREG 2000   LD    B,F0        06 F0
LOOP   2002   CALL  PRINTS      CD 0C 00
       2005   CALL  PRINTS      CD 0C 00
       2008   LD    A,R         ED 5F
       200A   CALL  OUTBYT      CD 30 AF
       200D   LD    A,10        3E 10        ;variieren
       200F   CALL  WARTEN      CD B8 AF
       2012   DJNZ  LOOP        10 EE
       2014   HALT              76
```

Experimentieren Sie mit anderen Verzögerungen und lassen Sie sie schließlich ganz fort.

Leider ist diese an sich schöne Möglichkeit nicht frei von Tücken: Das Refresh-Register zählt streng synchron zum Programmablauf. Zu einem bestimmten Programmabschnitt gehört auch eine ganz bestimmte Anzahl von Maschinen- bzw. Taktzyklen. Und während dieser bestimmten Anzahl von Taktzyklen zählt das Refresh-Register immer wieder um den gleichen Betrag weiter. Dadurch kann es geschehen, daß z. B. in einer Schleife bei jedem Durchgang an derselben Stelle stets derselbe R-Wert ausgelesen wird und sich der Zufallseffekt in sein gerades Gegenteil verkehrt.

Das folgende Programm lädt zum Experimentieren ein: TSTREF speichert die erhaltenen 160 Zufallszahlen nach 1200...129F ein.

RNDREF liest zunächst eine «ungültige» Randomzahl aus R aus und übernimmt sie nach B für eine einfache DJNZ-Verzögerung. Diese soll den Ablauf von Durchlauf zu Durchlauf unterschiedlich verlangsamen, damit R unterschiedlich viel Zeit zum Weiterzählen hat. RLC C hat hier nur den einzigen Sinn, den Ablauf um 8 Taktzyklen zu verlangsamen. Erst dann wird die «gültige» Zufallszahl ausgelesen.

```
P16.8.2  (Prüfsumme: 0CCF)

TSTREF 2000   LD    B,A0        06 A0
       2002   LD    HL,1200     21 00 12
LAB0   2005   CALL  RNDREF      CD 0F 20
       2008   LD    (HL),A      77
       2009   INC   HL          23
       200A   DJNZ  LAB0        10 F9
       200C   JP    DUMP12      C3 DF AF
```

```
RNDREF  200F  PUSH BC      C5
        2010  LD   A,R     ED 5F
        2012  LD   B,A     47
LAB1    2013  RLC  C       CB 01
        2015  DJNZ LAB1    10 FC
        2017  LD   A,R     ED 5F
        2019  POP  BC      C1
        201A  RET          C9
```

Überlegen Sie, weshalb dieser Algorithmus schon bald (im allgemeinen von der vierten Zahl an) zu konstanten Ergebnissen führt. Mit einigem Unglück können Sie aber auch Werte wie

08 48 48 48 48 48 48 48
48 48 48 48 . . . . . . erhalten.

A16.8.3 Wandeln Sie RNDREF so ab, daß Sie mit einer fest vorgegebenen Verzögerung experimentieren können. Verändern Sie diese Vorgabe. Vielleicht erhalten auch Sie einmal 00 01 02 03 04 ...?

A16.8.4 Stellen Sie fest, ob das R-Register inkrementiert oder dekrementiert wird. Nach je wie vielen Taktzyklen wird der Inhalt um eins verändert?

16.9 Vom Flag-Register

Wir haben z. B. in den Abschnitten 5.4 und 8.2 schon etwas über Flags gehört und sie auch schon mehrfach praktisch angewendet. Eine Systematik fehlte bisher; sie soll hier in Umrissen nachgeliefert werden und vor allem den Gebrauch des Z80-Handbuchs erleichtern. Wir haben bisher mit Carry-Flag und Zero-Flag gearbeitet.

Carry-Flag wird bei Addition und Subtraktion gesetzt, sobald ein Übertrag erfolgt. Es läßt sich mit SCF setzen und mit CCF komplementieren, außerdem durch AND, OR und XOR löschen. Auch die Vergleichsbefehle CP (compare) beeinflussen Carry-Flag: Ist beispielsweise A = 3E, so führt, ohne daß der Inhalt des Akkumulators dabei verändert wird, der Befehl CP 4A zur Subtraktion 3E − 4A, bei der ein Übertrag erfolgt und dementsprechend Carry gesetzt wird. Wenn Carry gesetzt ist, bewirken die Befehle ADC (SBC entsprechend) eine Addition nicht nur der beiden Operanden, sondern auch noch des Übertrags dazu.

Zero-Flag zeigt nach gewissen Operationen an, ob der Wert null erreicht wurde. Ist beispielsweise A = 34 und B = 34, so führt SUB A,B zur Subtraktion 34 − 34 = 00, und Zero-Flag wird gesetzt; bei CP B auch, jedoch ohne Veränderung des Akku-Inhalts.

Weitere Flags:
S-Flag (sign, Vorzeichen) wird gesetzt bei «negativem» Ergebnis. Als negativ in diesem Sinne wird eine Zahl angesehen, deren höchstes Bit gesetzt ist. Immer positiv sind die Zahlen 00...7F. Negativ in diesem Sinn sind die Zahlen 80...FF.
H-Flag zeigt einen Halbbyte-Übertrag (vom 3. zum 4. Bit) an. Es sorgt z. B. für den richtigen Ablauf des DAA-Befehls. Es steuert diese Dinge jedoch intern. Der Programmierer braucht es im allgemeinen nicht zu beachten.
N-Flag wirkt ebenfalls intern.
P/V-Flag hat doppelte Bedeutung: Als *P*arity-Bit wird es z. B. nach AND gesetzt, wenn die Zahl im Akku im Binärformat eine gerade Anzahl von Einsen enthält. Wir kommen darauf zurück. Als 0*V*erflow-Bit zeigt es einen Überlauf an. Bei 38 + 73 = AB = 1010 1011 wird beispielsweise als Summe zweier positiver Zahlen eine im obigen Sinne negative Zahl errechnet. Das könnte je nach Lesart zu einer Fehlinterpretation führen.
Alle diese Flags sind Bits im F-Register, dessen Inhalt bitweise zu deuten ist.

Eine sehr wesentliche Anwendung finden die Flags bei den bedingten Sprüngen. Suchen Sie bitte im Z80-Handbuch die Befehle JP c,nn, CALL c,nn und RET c, ferner die entsprechenden Befehle JR c,e.
c kann dabei bedeuten:

NZ = gleich null bzw. nicht gleich
Z = gleich null bzw. gleich
NC = kein Übertrag
C = Übertrag
PO = Parität ungerade (1, 3, 5, 7 Einsen)
PE = Parität gerade (0, 2, 4, 6, 8 Einsen)
P = positives Vorzeichen
M = negatives Vorzeichen («minus»)

Damit ist alles umrissen, was man beim praktischen Programmieren von den Flags wissen muß:

1. Sie werden beim Ablauf bestimmter Befehle automatisch gesetzt oder gelöscht. Nur Carry kann ohne Umweg willkürlich beeinflußt werden.
2. Die bedingten Sprungbefehle fragen automatisch die bedingenden Bits des Flag-Registers ab und werden nach ihrer Maßgabe ausgeführt oder ignoriert.
3. Darüber hinaus ist das F-Register für direkte Eingriffe nicht zugänglich.

Und gerade das letztere (3.) wollen wir jetzt tun, jedoch nur zur theoretischen Einblicknahme, nicht für den normalen praktischen Gebrauch. Den Lesezugriff erzwingen wir durch PUSH AF mit nachfolgendem POP DE, wodurch der Inhalt des F-Registers ins F-Register übertragen wird. Zum Schreiben verfahren wir umgekehrt. Damit konzipieren wir folgendes Rahmenprogramm:
1. Wir versetzten F (von E über Stack) in einen definierten Anfangszustand.
2. Wir lassen einige Speicherplätze für Testbefehle frei (NOPs).
3. Wir lesen F über Stack aus E aus und zeigen das Resultat binär an.

```
P16.9.1   (Prüfsumme: 0AAE)
TSTFLG 2000  LD   E,FF           1E FF          ;1.
       2002  PUSH DE             D5
       2003  POP  AF             F1
       2004  NOP NOP NOP NOP     00 00 00 00    ;2.
       2008  NOP NOP NOP NOP     00 00 00 00
       200C  PUSH AF             F5             ;3.
       200D  POP  DE             D1
       200E  LD   A,E            7B
       200F  CALL OUTBIN         CD F8 AE
       2012  CALL PAUSWK         CD D4 AF
       2015  RST  00             C7
```

Fügen wir nun

2006 AND A A7

ein, so wird u. a. Carry gelöscht. Wir erhalten F = 0001 0100, zwei Flags blieben also gesetzt. Aus dem Kommentar zu AND r geht hervor, daß eins davon das F-Flag ist; das andere könnte S-, Z- oder P/V-Flag sein. Probieren wir weiter, fügen hinzu

2004 LD A,7F 3E 7F

und variieren den Wert in 2005. In symbolischer Notierung erhalten wir dann die Ergebnisse:

3E 7F A7 → 0011 1000
3E 7E A7 → 0011 1100
3E 7D A7 → 0011 1100
3E 11 A7 → 0001 0100
3E 10 A7 → 0001 0000
3E 00 A7 → 0101 0100

Im letzten Beispiel verriet sich Zero-Flag als Bit6; es wird nach AND A gesetzt, wenn der Inhalt des Akkumulators 00 ist.

Wir zählen nun die Anzahl der Einsen in der Binärdarstellung unserer Operanden:

Anzl(7F) = 7 Anzl(7D) = 6 Anzl(10) = 1
Anzl(7E) = 6 Anzl(11) = 2 Anzl(00) = 0

Wir beobachten, daß immer, wenn eine gerade Anzahl von Einsen vorliegt, Bit2 im Flag-Register gesetzt ist. Es ist offenbar – in diesem Zusammenhang – das Paritätsbit, das «parity even», also eine gerade Anzahl von Einsen anzeigt.

Nutzanwendung 1:

Der ASCII ist von Haus aus ein 7-bit-Code; die zu codierenden Zeichen werden in den Bits6...0 dargestellt. Bit7 bleibt zunächst unbenutzt, wird dann aber manchmal zur Kontrolle einer korrekten Datenübertragung herangezogen und so belegt, daß insgesamt eine gerade Anzahl von Einsen vorliegt. Wird bei einer Kontrolle P/V = 0 festgestellt, liegt ein Lese- oder Interpretationsfehler vor, dem nachzugehen ist.

A16.9.2 Fügen Sie in P16.9.1 andere Operationen ein, und versuchen Sie, andere Flags zu identifizieren.

A16.9.3 Schreiben Sie ein Programm, das nach Eingabe einer zweistelligen Hexadezimalzahl die Anzahl der Einsen in ihrer Binärdarstellung ermittelt und anzeigt!

Nutzanwendung 2:

Der Befehl DJNZ dekrementiert das B-Register. Dies läßt als 8-Bit-Register maximal 256 Schleifen zu. Der Befehl CPD (und andere, siehe Handbuch) dekrementiert das BC-Register. Dieses läßt als 16-Bit-Register maximal 65536 Durchläufe zu. Wenn BC-0 erreicht ist, wird das PV-Flag – ziemlich zweckentfremdet – gelöscht.

CPD ist ein Blocksuchbefehl. Wir werden ihn als solchen nicht besprechen, sondern benutzen ihn nur als Schleifenzähler.

```
P16.9.4  3000  DD 21 00 00   LD    IX,0000
         3004  01 34 12      LD    BC,1234
LABEL    3007  DD 23         INC   IX
         3009  ED A9         CPD
         300B  EA 07 30      JP    PE,LABEL
         300E  DD 22 00 12   LD    (1200),IX
         3012  C3 DF AF      JP    DUMP12
```

Während hier das BC-Register für jeden Durchlauf dekrementiert wird, wird – von null angefangen – das IX-Register je Schleifendurchlauf einmal inkrementiert und zur Kontrolle nach 1200 (+01) abgelegt. Sie könnten versuchen, IX je Schleife zweimal (oder dreimal) zu inkrementieren.

17
Permutationen

Nehmen wir an, wir hätten eine Grundmenge von drei Elementen, z. B. 0, 1, 2. Anders als in der Mengenlehre soll es jedoch gerade auf die Reihenfolge der Elemente ankommen. Dann sind sechs verschiedene Anordnungen möglich:

012 021 102 120 201 210

Man nennt sie Permutationen.

Von 2 Elementen gibt es 2 Permutationen
von 3 Elementen gibt es 6 Permutationen
von 4 Elementen gibt es 24 Permutationen
von 5 Elementen gibt es 120 Permutationen

von 10 Elementen gibt es 3 628 800 Permutationen usw. Es gibt nun Aufgaben, die man – häufig sinnvoller – durch Überlegen, manchmal aber auch durch schematisches Ausprobieren lösen kann. Sehr schöne Aufgaben dieser Art sind Zahlenrätsel wie z. B.

KIND	mit	8270
+ BALL	der	+ 5966
= SPIEL	Lösung	= 14236,

bei denen jeder Buchstabe (unter Berücksichtigung des jeweiligen Stellenwerts) eine Variable darstellt, die für eine der Ziffern des Dezimalsystems steht. Die Lösung durch logisches Schließen kann ein Leckerbissen für Freunde des Denksports sein, aber auch das Programmieren macht Spaß.

Ein Computer ist ein schneller, zuverlässiger und belastbarer Rechenknecht, der in annehmbarer Zeit alle 3 628 800 möglichen Zuordnungen der 10 Ziffern zu 10 Variablen durchprüft. Wie aber erzeugt man diese 3,63 Millionen Zuordnungen (Permutationen)?

17.1 Emporzählen mit Disqualifikation

Wir versuchen zunächst einen naheliegenden, wenn auch recht primitiven Weg, indem wir in 5 oder 10 Speicherstellen von 0000000000 bis 9999999999 zählen lassen und dabei prüfen, ob unzulässige oder doppelte Ziffern entstanden sind. Wenn ja, dann lassen wir ohne Beachtung weiterzählen; wenn nein, dann haben wir eine (neue) Permutation erreicht, mit der die Rechnung zu überprüfen wäre.

Um den Überblick zu behalten, beschränken wir uns auf die Ziffern 0, 1, 2, 3 und legen sie in den Speicherstellen 1201...1204 ab. 1200 dient als Überlaufanzeiger. Tritt hier ein Wert > 0 auf, so ist unser Zähler übergelaufen – Schluß!

Grobstruktur:
1. 1200... auf null setzen.
2. HL auf die Ziffer ganz rechts setzen und Überlauf abfragen.
3. (HL) inkrementieren.
4. Wenn (HL) 〈 4, dann 6.
5. (HL) = 0, HL = HL − 1, zurück nach 3.
6. Wenn gleiche Ziffern, dann zurück nach 2.
7. Ausgeben und zurück nach 2.

Feinstruktur:
6a HL neben die Ziffer ganz rechts setzen.
6b HL dekrementieren.
6c Wenn L = 1, dann Ausgabe 7.
6d DE = HL.
6e DE dekrementieren.
6f Wenn E = 0, dann 6b.
6g Wenn (HL) = (DE), dann zurück nach 2.
6h Zurück nach 6e

```
P17.1.1   (Prüfsumme: 19F0)

PERMUT 3000  CALL  NULL12        CD F4 AF  ;1.
L1     3003  LD    HL,1204       21 04 12  ;2.
       3006  LD    A,(1200)      3A 00 12
       3009  AND   A             A7
       300A  JR    Z,L2          28 04
       300C  CALL  PAUSWK        CD D4 AF
       300F  RST   00            C7
```

```
L2      3010  INC   (HL)       34          ;3.
        3011  LD    A,(HL)     7E
        3012  CP    04         FE 04
        3014  JR    NZ,GLEIZI  20 18       ;4.
        3016  LD    (HL),00    36 00       ;5.
        3018  DEC   HL         2B
        3019  JR    L2         18 F5
AUSGEB  301B  LD    B,04       06 04       ;7.
        301D  LD    HL,1201    21 01 12
        3020  CALL  LETNL      CD 06 00
L3      3023  LD    A,(HL)     7E
        3024  OR    30         F6 30
        3026  CALL  PRNT       CD 12 00
        3029  INC   HL         23
        302A  DJNZ  L3         10 F7
        302C  JR    L1         18 D5
GLEIZI  302E  LD    HL,1205    21 05 12    ;6a
L4      3031  DEC   HL         2B          ;6b
        3032  LD    A,L        7D          ;6c
        3033  CP    01         FE 01
        3035  JR    Z,AUSGEB   28 E4
        3037  PUSH  HL         E5          ;6d
        3038  POP   DE         D1
L5      3039  DEC   DE         1B          ;6e
        303A  LD    A,E        7B
        303B  AND   A          A7          ;6f
        303C  JR    Z,L4       28 F3
        303E  LD    C,(HL)     4E
        303F  LD    A,(DE)     1A
        3040  CP    C          B9          ;6g
        3041  JR    Z,L1       28 C0
        3043  JR    L5         18 F4       ;6h
```

A17.1.2
1. Welche Speicherstellen müssen verändert werden, um 5, 6, 7, ... Elemente zu permutieren?
2. Ermitteln Sie, wieviel Zeit zwischen Programmstart und erster Ausgabe bei 4, 5, 6, ... Elementen vergeht!

P17.1.1 ist zur Lösung eines Zahlenrätsels ungeeignet. Man könnte den Vorlauf zwar deutlich verkürzen, indem man schon von 0123456788 aus startet. Die zeitraubenden Zwischenläufe blieben aber trotzdem.

17.2 Schnelle Zufallspermutationen

Nach dem teilweise äußerst langsamen P17.1.1 schreiben wir nun ein Programm, das sehr schnell ist, die Permutationen in bunter Reihenfolge liefert und mehrfaches Hintereinanderauftreten der gleichen Permutation zuläßt. Für strenge statistische Untersuchungen wäre dieses Konzept kaum zu gebrauchen; für Spiele mit Überraschungseffekten ist es jedoch bestens geeignet.

Speicherdisposition: In den zehn Speicherstellen 1200 ... 1209 stehen die zehn Ziffern 0 ... 9 und werden hier je zwei und zwei vertauscht.

Grobstruktur:
1. 1200... auf null setzen und Uhr starten.
2. Ziffern 0...9 einschreiben.
3. HL auf Zufallswert 1200 ≦ Zfw ≦ 1200 setzen.
4. DE dto.
5. (HL) und (DE) vertauschen; Zwischenspeicher: C.
6. Anzeigen und Ablauf momentan unterbrechen.
7. Zurück nach 3.

(8.) Randomroutine.

```
P17.2.1  (Prüfsumme: 14F2)

ZFPERM 2800  CALL NULL12     CD F4 AF ;1.
       2803  XOR  A          AF
       2804  LD   DE,1200    11 00 12
       2807  CALL TIMST      CD 33 00
       280A  XOR  A          AF       ;2.
       280B  LD   B,0A       06 0A
       280D  LD   HL,1200    21 00 12
L1     2810  LD   (HL),A     77
       2811  INC  A          3C
       2812  INC  HL         23
       2813  DJNZ L1         10 FB
LOOP   2815  LD   H,12       26 12    ;3.
       2817  CALL RND        CD 3B 28
       281A  LD   L,A        6F
       281B  LD   D,H        54       ;4.
       281C  CALL RND        CD 3B 28
       281F  LD   E,A        5F
       2820  LD   C,(HL)     4E       ;5.
       2821  LD   A,(DE)     1A
```

```
        2822 LD   (HL),A     77
        2823 LD   A,C        79
        2824 LD   (DE),A     12
AUSGEB  2825 LD   HL,1200    21 00 12 ;6.
        2828 LD   B,0A       06 0A
        282A CALL LETNL      CD 06 00
L2      282D LD   A,(HL)     7E
        282E OR   30         F6 30
        2830 CALL PRNT       CD 12 00
        2833 INC  HL         23
        2834 DJNZ L2         10 F7
        2836 NOP NOP NOP     00 00 00
        2839 JR   LOOP       18 DA    ;7.
RND     283B LD   A,(E005)   3A 05 E0 ;8.
        283E DAA             27
        283F AND  0F         E6 0F
        2841 RET             C9
```

Der Ablauf zeigt einen eventuellen Mangel und einen Fehler:
1. Das Programm läuft so schnell ab, daß man die gezeigten Permutationen kaum kontrollieren kann; es wurde jedoch Platz für PAUSKY oder PAUSWK gelassen.
2. Schon bald treten fehlerhafte Ausgaben auf. Die Fehlersuche (debugging) nimmt in der Praxis des Programmierens einen breiten Raum ein. Wichtig ist dabei, sich nicht ins Bockshorn jagen zu lassen, sondern einen kühlen Kopf zu bewahren. Das Ausgabeformat entspricht der Absicht; an der Ausgabe liegt es also wohl nicht. Stimmen denn aber die auszugebenden Werte? D1200 liefert vereinfacht folgendes Bild:

```
1200 00 01 02 00 04 05 06 07
1208 08 09 00 00 03 00 00 00
1210 00 00 00 00 00 00 00 00
```

Hier wurden (1203) und (120C) vertauscht. Der Tausch selbst erfolgt korrekt, doch wurde eine unzulässige Tauschadresse erzeugt; genauer: ihr LSB ist falsch. Das aber wird in der RND-Routine erzeugt; also müssen wir hier den Fehler suchen. Die Zeilen 283B, 283F und 2841 sind über jeden Zweifel erhaben. Wozu eigentlich das DAA? Es soll nichtdezimale Ziffern in dezimale umwandeln, also – für das ganze Byte beschrieben – beispielsweise 4C in 52. Offenbar blieb aber xC erhalten und erzeugte die falsche Adresse 120C. DAA funktionierte

```
A B C D * (2,3)  }P23   }P13   }P03
A B D C * (2,3)  }      }      }
A B C D   (1,2)  ----   }      }
A C B D * (2,3)  }P23   }      }
A C D B * (2,3)  }      }      }
A C B D   (1,3)  ----   }      }
A D B C * (2,3)  }P23   }      }
A D C B * (2,3)  }      }      }
A D B C   (1,2)  ----   }      }
A B D C   (2,3)         }      }
A B C D   (0,1)  -----------   }
B A C D * (2,3)  }P23   }P13   }
B A D C * (2,3)  }      }      }
B A C D   (1,2)  ----   }      }
B C A D * (2,3)  }P23   }      }
B C D A * (2,3)  }      }      }
B C A D   (1,3)  ----   }      }
B D A C * (2,3)  }P23   }      }
B D C A * (2,3)  }      }      }
B D A C   (1,2)  ----   }      }
B A D C   (2,3)         }      }
B A C D   (0,2)  -----------   }
C A B D * (2,3)  }P23   }P13   }
C A D B * (2,3)  }      }      }
C A B D   (1,2)  ----   }      }
C B A D * (2,3)  }P23   }      }
C B D A * (2,3)  }      }      }
C B A D   (1,3)  ----   }      }
C D A B * (2,3)  }P23   }      }
C D B A * (2,3)  }      }      }
C D A B   (1,2)  ----   }      }
C A D B   (2,3)         }      }
C A B D   (0,3)  -----------   }
D A B C * (2,3)  }P23   }P13   }
D A C B * (2,3)  }      }      }
D A B C   (1,2)  ----   }      }
D B A C * (2,3)  }P23   }      }
D B C A * (2,3)  }      }      }
D B A C   (1,3)  ----   }      }
D C A B * (2,3)  }P23   }      }
D C B A * (2,3)  }      }      }
D C A B   (1,2)  ----   }      }
D A C B   (2,3)         }      }
D A B C   (0,1)  -----------   }
A D B C   (1,2)                }
A B D C   (2,3)                }
A B C D   Schluß
```

Bild 17.1 Ein Permutationsalgorithmus

also nicht wie erwartet. Hier erinnern wir uns an die Bemerkung zum H-Flag (Abschnitt 16.8): Es beeinflußt den Ablauf von DAA. Wir müssen also noch eine Operation einfügen, die die Flags in den richtigen definierten Zustand versetzt. Als solche bietet sich ADD 00 an:

```
P17.2.2
RND     283B  LD    A,(E005)     3A 05 E0
        283E  ADD   A,00         C6 00
        2840  DAA                27
        2841  AND   0F           E6 0F
        2843  RET                C9
```

Schon ist der Schaden behoben.

17.3 Systematisches Permutieren

Nach den beiden Vorläufern P17.1.1 und P17.2.1 entwerfen wir nun ein Programm, das recht schnell ist, jede Permutation nur einmal auswirft – und das in geordneter Reihenfolge. Da der zugrunde liegende Algorithmus nicht auf den ersten Blick erkennbar ist, holen wir etwas weiter aus. Wegen der Länge des Programms beschränken wir uns bei der ersten Verständigungsskizze auf vier Elemente: A, B, C und D. Die zugehörigen Speicherplätze nennen wir kurz, 0, 1, 2, 3 und meinen damit 1200 ... 1203. Ein Sternchen in Bild 17.1 soll «Ausgabe» bedeuten und (1,3) den Austausch von (1201) und (1203). Pxy ist die Kurzbezeichnung eines Programmabschnitts, der das x. bis y. Element permutiert.

Wir übersetzen diesen Algorithmus nahezu wörtlich in die Maschinensprache. Dabei definieren wir die Vertauschungen im BC-Register und tauschen analog zu P17.2.1 (2815 ... 2824).

```
P17.3.1   (Prüfsummen am rechten Rand)

PERM03  4000  LD    A,41         3E 41
        4002  LD    B,04         06 04
        4004  LD    HL,1200      21 00 12
L1      4007  LD    (HL),A       77
        4008  INC   A            3C
        4009  INC   HL           23
```

```
        400A   DJNZ L1         10 FB
        400C   CALL P03        CD 48 40
        400F   CALL PAUSWK     CD D4 AF
        4012   RST  00         C7                 ;0709
P23     4013   CALL AUSGEB     CD 84 40
        4016   LD   BC,0203    01 03 02
        4019   CALL VERTAU     CD 79 40
        401C   CALL AUSGEB     CD 84 40
        401F   LD   BC,0203    01 03 02
        4022   CALL VERTAU     CD 79 40
        4025   RET             C9                 ;0E0C
P13     4026   CALL P23        CD 13 40
        4029   LD   BC,0102    01 02 01
        402C   CALL VERTAU     CD 79 40
        402F   CALL P23        CD 13 40
        4032   LD   BC,0103    01 03 01
        4035   CALL VERTAU     CD 79 40
        4038   CALL P23        CD 13 40
        403B   LD   BC,0102    01 02 01
        403E   CALL VERTAU     CD 79 40
        4041   LD   BC,0203    01 03 02
        4044   CALL VERTAU     CD 79 40
        4047   RET             C9                 ;1860
P03     4048   CALL P13        CD 26 40
        404B   LD   BC,0001    01 01 00
        404E   CALL VERTAU     CD 79 40
        4051   CALL P13        CD 26 40
        4054   LD   BC,0002    01 02 00
        4057   CALL VERTAU     CD 79 40
        405A   CALL P13        CD 26 40
        405D   LD   BC,0003    01 03 00
        4060   CALL VERTAU     CD 79 40
        4063   CALL P13        CD 26 40
        4066   LD   BC,0001    01 01 00
        4069   CALL VERTAU     CD 79 40
        406C   LD   BC,0102    01 02 01
        406F   CALL VERTAU     CD 79 40
        4072   LD   BC,0203    01 03 02
        4075   CALL VERTAU     CD 79 40
        4078   RET             C9                 ;272E
VERTAU  4079   LD   H,12       26 12
        407B   LD   D,H        54
        407C   LD   L,B        68
        407D   LD   E,C        59
        407E   LD   C,(HL)     4E
        407F   LD   A,(DE)     1A
```

```
        4080  LD    (HL),A     77
        4081  LD    A,C        79
        4082  LD    (DE),A     12
        4083  RET              C9          ;2AAE
AUSGEB  4084  CALL  LETNL      CD 06 00
        4087  LD    B,04       06 04
        4089  LD    HL,1200    21 00 12
L2      408C  LD    A,(HL)     7E
        408D  CALL  PRNT       CD 12 00
        4090  INC   HL         23
        4091  DJNZ  L2         10 F9
        4093  RET              C9          ;2F10
```

Das Programm wurde übersichtlichkeitshalber nicht auf die kürzestmögliche Form gebracht. Versuchen Sie, es in folgenden Schritten zu verstehen:

1. Vollziehen Sie das Vertauschungsschema Schritt für Schritt nach.
2. Erkennen Sie das Vertauschungsschema Schritt für Schritt im Programm wieder.
3. Ändern Sie das Programm wie folgt ab:
   a) Die Elemente U, V, W, X oder 1, 2, 3, 4 sollen permutiert werden.
   b) Speichern Sie manuell die Codes für «?», «!», «+» und «*» ein und starten Sie durch Einsprung an geeigneter Stelle.
4. Überlegen Sie, wie das Programm zur Permutation von fünf Elementen zu verändern wäre.

Dieser Ansatz kann erweitert werden. Er ist wohl der schnellste Lieferant geordneter Permutationen. Leider stellt er erhebliche Anforderungen an den Fleiß, so daß wir einer interessanteren Lösung nachgehen:

17.4 Permutieren durch Rekursion GAUSS + RIESE = EUKLID

Dieses schöne Zahlenrätsel wurde vor einiger Zeit in der Sendung «Kopf um Kopf» vorgestellt, ist aber auch in der einschlägigen Knobelliteratur zu finden. Es ist nicht immer sinnvoll, den Computer überall da einzusetzen, wo man ihn einsetzen kann. Hier sollte man sich nicht um die reizvolle Denkaufgabe betrügen; aber das Programmieren ist ja auch Denksport!

Rüdeger Baumann stellt in [5] auf Seite 153 ff. einen Algorithmus zur rekursiven Erzeugung von Permutationen vor. Rekursive (Unter-)Programme rufen sich selber auf, natürlich gut kontrolliert, damit sich der Ablauf nicht im Unendlichen verliert. Die Programmiersprache Pascal läßt Rekursionen zu, BASIC im allgemeinen nicht. Das S-BASIC unseres MZ-700 macht hier eine Ausnahme. Es erlaubt, in FOR-TO-NEXT-Loops Feldvariable als Laufvariable zu verwenden. Damit läßt sich der o. a. Algorithmus in BASIC übersetzen:

```
10 REM *** P17.4.1 ***
20 CLS: PRINT"PERMUTATIONEN": PRINT
30 INPUT"Wie viele Elemente? ";N: K=N-1
40 DIM Z(K),I(K)
50 FOR I=0 TO K: Z(I)=I: NEXT I
60 GOSUB 1000
70 GOTO 70
80 :
1000 IF K=0 THEN GOSUB 3000: RETURN
1010 K=K-1: GOSUB 1000: K=K+1
1020 FOR I(K)=0 TO K-1
1030 GOSUB 2000
1040 K=K-1: GOSUB 1000: K=K+1
1050 GOSUB 2000
1060 NEXT I(K)
1070 RETURN
1080 :
2000 Z=Z(I(K)): Z(I(K))=Z(K): Z(K)=Z
2010 RETURN
3000 FOR I=0 TO N-1: PRINTZ(I);: NEXT I
3010 PRINT: RETURN
```

Wir gehen auf die Wirkungsweise nicht im einzelnen ein und stellen nur fest, daß dieses Programm die ineinander verschachtelten Vertauschungen korrekt nachvollzieht, wenn auch nicht in der «geordneten» Reihenfolge von P17.3.1. Wir übersetzen es «wörtlich» in die Maschinensprache und wenden es gleich auf das oben genannte Zahlenrätsel an.

```
P17.4.2   (Prüfsummen beginnen hier mit "*")
001 0000               ;GAUSS + RIESE = EUKLID
002 0000               ;01.09.84 - 17.30
003 0000               ;
004 0000               LETNL:  EQU  0006
005 0000               PRINTS: EQU  000CH
006 0000               PRNT:   EQU  0012H
007 0000               MSG:    EQU  0015H
008 0000               GETKY:  EQU  001BH
009 0000               BELL:   EQU  003EH
010 0000               ;
011 0000               Z0:     EQU  1200H       ;Z0...Z9
012 0000               CARRY:  EQU  Z0+10
013 0000               NULL:   EQU  CARRY+01
014 0000               K:      EQU  NULL+01
015 0000               I0:     EQU  K+01        ;I0...I9
016 0000               Z:      EQU  I0+10
017 0000               ADIK:   EQU  Z+01
018 0000               ADZK:   EQU  ADIK+02
019 0000               ADZIK:  EQU  ADZK+02
020 0000               ;
021 0000                       REL  4000H
022 4000               ;
023 4000 CD1C40        START:  CALL GARIEU
024 4003 CD3840                CALL INIT
025 4006 CD4C40                CALL PERM
026 4009 11CF41                LD   DE,TABEND
027 400C CD1500                CALL MSG
028 400F 0620                  LD   B,20H
029 4011 CD3E00        LS0:    CALL BELL
030 4014 10FB                  DJNZ LS0
031 4016 CD1B00        LS1:    CALL GETKY
032 4019 28FB                  JR   Z,LS1
033 401B C7                    RST  00          ;*0AD8
034 401C               ;
035 401C 11B641        GARIEU: LD   DE,TABGAU
036 401F CD1500                CALL MSG
```

```
037 4022 CD0600              CALL LETNL
038 4025 11BF41              LD   DE,TABRIE
039 4028 CD1500              CALL MSG
040 402B CD0600              CALL LETNL
041 402E 11C741              LD   DE,TABEUK
042 4031 CD1500              CALL MSG
043 4034 CD0600              CALL LETNL
044 4037 C9                  RET                 ;*13F2
045 4038              ;
046 4038 3E09         INIT:  LD   A,09
047 403A 320C12              LD   (K),A
048 403D AF                  XOR  A
049 403E 320B12              LD   (NULL),A
050 4041 060A                LD   B,0AH
051 4043 210012              LD   HL,Z0
052 4046 77           LIN0:  LD   (HL),A
053 4047 3C                  INC  A
054 4048 23                  INC  HL
055 4049 10FB                DJNZ LIN0
056 404B C9                  RET                 ;*1874
057 404C              ;
058 404C 3A0C12       PERM:  LD   A,(K)
059 404F A7                  AND  A
060 4050 C25740              JP   NZ,LP0
061 4053 CD8940              CALL PRUEF
062 4056 C9                  RET
063 4057 210C12       LP0:   LD   HL,K
064 405A 35                  DEC  (HL)
065 405B CD4C40              CALL PERM
066 405E 210C12              LD   HL,K
067 4061 34                  INC  (HL)
068 4062 CD7541              CALL SADIK
069 4065 2A1812              LD   HL,(ADIK)
070 4068 3600                LD   (HL),00        ;*2178
071 406A CD9E41       LP1:   CALL TAUSCH
072 406D 210C12              LD   HL,K
073 4070 35                  DEC  (HL)
074 4071 CD4C40              CALL PERM
075 4074 210C12              LD   HL,K
076 4077 34                  INC  (HL)
077 4078 CD9E41              CALL TAUSCH
078 407B CD7541              CALL SADIK
079 407E 2A1812              LD   HL,(ADIK)
080 4081 34                  INC  (HL)
081 4082 3A0C12              LD   A,(K)
082 4085 BE                  CP   (HL)
```

```
083 4086 20E2              JR   NZ,LP1
084 4088 C9                RET                ;*2BFC
085 4089          ;
086 4089 AF       PRUEF:   XOR  A
087 408A 320A12            LD   (CARRY),A
088 408D 012108            LD   BC,0821H
089 4090 CDB440            CALL PRUEF2
090 4093 018408            LD   BC,0884H
091 4096 CDB440            CALL PRUEF2
092 4099 012609            LD   BC,0926H
093 409C CDB440            CALL PRUEF2
094 409F 014500            LD   BC,0045H
095 40A2 CDB440            CALL PRUEF2
096 40A5 017903            LD   BC,0379H
097 40A8 CDB440            CALL PRUEF2
098 40AB 01B20B            LD   BC,0BB2H   ;ZB=NULL
099 40AE CDB440            CALL PRUEF2
100 40B1 C3EF40            JP   AUSGEB        ;*3BD9
101 40B4 78       PRUEF2:  LD   A,B
102 40B5 32D340            LD   (LP20+02),A
103 40B8 79                LD   A,C
104 40B9 CB3F              SRL  A
105 40BB CB3F              SRL  A
106 40BD CB3F              SRL  A
107 40BF CB3F              SRL  A
108 40C1 32D640            LD   (LP21+02),A
109 40C4 79                LD   A,C
110 40C5 E60F              AND  0FH
111 40C7 32DD40            LD   (LP22+02),A
112 40CA DD210012          LD   IX,Z0
113 40CE 3A0A12            LD   A,(CARRY)     ;*47A2
114 40D1 DD8600   LP20:    ADD  A,(IX+00)
115 40D4 DD8600   LP21:    ADD  A,(IX+00)
116 40D7 27                DAA
117 40D8 4F                LD   C,A
118 40D9 E60F              AND  0FH
119 40DB DDBE00   LP22:    CP   (IX+00)
120 40DE 200D              JR   NZ,AUS
121 40E0 79                LD   A,C
122 40E1 CB3F              SRL  A
123 40E3 CB3F              SRL  A
124 40E5 CB3F              SRL  A
125 40E7 CB3F              SRL  A
126 40E9 320A12            LD   (CARRY),A
127 40EC C9                RET
128 40ED F1       AUS:     POP  AF
```

```
129 40EE C9                  RET              ;*550D
130 40EF            ;
131 40EF CD0600     AUSGEB:  CALL LETNL
132 40F2 CD0C00              CALL PRINTS
133 40F5 CD0C00              CALL PRINTS
134 40F8 3A0312              LD   A,(Z0+03)   ;=(Z3)
135 40FB CD6F41              CALL PRNT2
136 40FE 3A0012              LD   A,(Z0)      ;=(Z0)
137 4101 CD6F41              CALL PRNT2
138 4104 3A0912              LD   A,(Z0+09)   ;=(Z9)
139 4107 CD6F41              CALL PRNT2
140 410A 3A0812              LD   A,(Z0+08)   ;=(Z8)
141 410D CD6F41              CALL PRNT2
142 4110 3A0812              LD   A,(Z0+08)   ;=(Z8)
143 4113 CD6F41              CALL PRNT2
144 4116 CD0600              CALL LETNL
145 4119 3E2B                LD   A,2BH       ;'+'
146 411B CD1200              CALL PRNT
147 411E CD0C00              CALL PRINTS
148 4121 3A0712              LD   A,(Z0+07)   ;=(Z7)
149 4124 CD6F41              CALL PRNT2
150 4127 3A0412              LD   A,(Z0+04)   ;=(Z4)
151 412A CD6F41              CALL PRNT2
152 412D 3A0212              LD   A,(Z0+02)   ;*677A
153 4130 CD6F41              CALL PRNT2
154 4133 3A0812              LD   A,(Z0+08)   ;=(Z8)
155 4136 CD6F41              CALL PRNT2
156 4139 3A0212              LD   A,(Z0+02)   ;=(Z2)
157 413C CD6F41              CALL PRNT2
158 413F CD0600              CALL LETNL
159 4142 3E3D                LD   A,3DH       ;'='
160 4144 CD1200              CALL PRNT
161 4147 3A0212              LD   A,(Z0+02)   ;=(Z2)
162 414A CD6F41              CALL PRNT2
163 414D 3A0912              LD   A,(Z0+09)   ;=(Z9)
164 4150 CD6F41              CALL PRNT2
165 4153 3A0512              LD   A,(Z0+05)   ;=(Z5)
166 4156 CD6F41              CALL PRNT2
167 4159 3A0612              LD   A,(Z0+06)   ;=(Z6)
168 415C CD6F41              CALL PRNT2
169 415F 3A0412              LD   A,(Z0+04)   ;=(Z4)
170 4162 CD6F41              CALL PRNT2
171 4165 3A0112              LD   A,(Z0+01)   ;=(Z1)
172 4168 CD6F41              CALL PRNT2
173 416B CD0600              CALL LETNL
174 416E C9                  RET
```

```
175 416F F630       PRNT2:  OR   30H
176 4171 CD1200             CALL PRNT
177 4174 C9                 RET                    ;*7DFB
178 4175            ;
179 4175 210D12     SADIK:  LD   HL,I0
180 4178 3A0C12             LD   A,(K)
181 417B 85                 ADD  A,L
182 417C 6F                 LD   L,A
183 417D 221812             LD   (ADIK),HL
184 4180 C9                 RET
185 4181 210012     SADZK:  LD   HL,Z0
186 4184 3A0C12             LD   A,(K)
187 4187 85                 ADD  A,L
188 4188 6F                 LD   L,A
189 4189 221A12             LD   (ADZK),HL
190 418C C9                 RET
191 418D CD7541     SADZIK: CALL SADIK
192 4190 210012             LD   HL,Z0
193 4193 ED5B1812           LD   DE,(ADIK)
194 4197 1A                 LD   A,(DE)
195 4198 85                 ADD  A,L
196 4199 6F                 LD   L,A
197 419A 221C12             LD   (ADZIK),HL
198 419D C9                 RET                    ;*8881
199 419E            ;
200 419E CD8D41     TAUSCH  CALL SADZIK
201 41A1 CD8141             CALL SADZK
202 41A4 2A1C12             LD   HL,(ADZIK)
203 41A7 ED5B1A12           LD   DE,(ADZK)
204 41AB 7E                 LD   A,(HL)
205 41AC 321712             LD   (Z),A
206 41AF 1A                 LD   A,(DE)
207 41B0 77                 LD   (HL),A
208 41B1 3A1712             LD   A,(Z)
209 41B4 12                 LD   (DE),A
210 41B5 C9                 RET                    ;*901F
211 41B6            ;
212 41B6 16         TABGAU: DEFB 16H
213 41B7 20204741           DEFM '  GAUSS'
214 41BB 555353
215 41BE 0D                 DEFB 0DH
216 41BF 2B205249   TABRIE: DEFM '+ RIESE'
217 41C3 455345
218 41C6 0D                 DEFB 0DH
219 41C7 3D45554B   TABEUK: DEFM '=EUKLID'
220 41CB 4C4944
221 41CE 0D                 DEFB 0DH
```

```
222 41CF 454E4445  TABEND: DEFM 'ENDE'
223 41D3 0D                DEFB 0DH          ;*9706
224 41D4           ;
225 41D4                   END
```

1200...1209	Z0...Z9 zu permutierende Ziffern	((011))
120A	CARRY ist eine Carry-Simulation	((012))
120B	NULL zur einheitlichen Addition	((013))
120C	K: Verschachtelungstiefe	((014))
120D...1216	I0...I9: Laufvariable I(K)	((015))
1217	Z: dritte Variable zum Tauschen	((016))
1218...1219	ADIK: Adresse von I(K)	((017))
121A...121B	ADZK: Adresse von Z(K)	((018))
121C...121D	ADZIK: Adresse von Z(I(K))	((019))

START: Das Hauptprogramm hat nur eine wesentliche Aufgabe: Es ruft das Unterprogramm PERMutieren auf. Alles andere ist lediglich Koordination von leicht verständlichen Nebenaufgaben, die bei den jeweiligen Programmabschnitten erläutert werden. ((023))

GARIEU: Schreibt die Namen der drei Mathematiker in Form einer Addition untereinander auf den Bildschirm. ((035))

INIT: Initialisiert die Variablen. PERM beginnt in der 9. Ebene. NULL erhält den Wert null. Den zu permutierenden Elementen Z0...Z9 werden die Anfangswerte zugewiesen. ((046))

PERM: ist eine nahezu wörtliche Übersetzung der BASIC-Programmzeilen 1000...1070. Anders als dort werden hier nur die Permutationen ausgegeben, die das Zahlenrätsel erfüllen. ((058))

PRUEF: setzt zuerst das simulierte Carry auf null. Dann prüft es, ob die zuerst auszuführende Teiladdition S + E = D erfüllt ist. Die Buchstaben der drei Namen sind den Variablen Zi wie folgt zugeordnet: ((086))

A	D	E	G	I	K	L	R	S	U
Z0	Z1	Z2	Z3	Z4	Z5	Z6	Z7	Z8	Z9

LD BC,0821H codiert die Teiladdition Z8 + Z2 = Z1 bzw. S + E = D.

PRUEF2: führt die eigentliche Prüfung durch. Das B-Register enthält den Index des obersten Summanden; er wird als Offset zu IX nach 40D3 eingespeichert. Das höherwertige Nibble des C-Registers enthält den Index des mittleren Summanden, der als Offset nach 40D6 geschickt wird, und das niederwertige Nibble den Index der jeweiligen Ziffer der Teilsumme. Stellt sich nach dem Maskieren mit 0F Ungleichheit ein, wird die Prüfung sofort abgebrochen, denn alle weiteren Teilprüfungen wären Zeitvergeudung. Bei Gleichheit wird die nach C zwischengespeicherte unveränderte Teilsumme zurückgeholt und um 4 Bit nach rechts geshiftet. Danach enthält der Akku entweder «0» oder – bei Übertrag – «1». Diese Zahl wird im Pseudo-Carry aufbewahrt. ((101))

AUSGEB: Durchläuft eine Permutation alle Prüfungen erfolgreich, so ist sie als Lösung aufbereitet auszugeben. Zu Zeile 134 mache man sich klar, daß man Z3 erreicht, indem man im Speicher von Z0 aus drei Stellen weitergeht. Z0+03 steht also für Z3, genauer: für den Speicherplatz, in den der Wert der Variablen Z3 eingeschrieben ist. ((131))

SADIK: Setzt bzw. berechnet die Adresse von I(K) und legt sie nach 1218 (+1) ab. ((179))

SADZK: Berechnet die Adresse von Z(K) und legt sie nach 121A (+1) ab. ((185))

SADZIK: Berechnet die Adresse von Z(I(K)) und legt sie nach 121C (+1) ab. ((191))

TAUSCH: ist sozusagen identisch mit Zeile 2000 des BASIC-Programms. ((200))

Daß dieses Programm schon in zweieinhalb Minuten alle Lösungen des Zahlenrätsels gefunden hat, ist kein Maß für seine Güte, denn es ist purer Zufall, daß die Lösungen in der hier gelieferten Reihenfolge der Permutationen schon recht früh auftreten. Aussagekräftiger ist die Gesamtlaufzeit. Sie beträgt hier 24 min gegenüber 9 min bei der am Ende von Abschnitt 17.3 angedeuteten Lösung. Dafür ist P17.4.2 mit 468 Bytes deutlich kürzer als jene Lösung (871 Bytes) und im Ansatz wesentlich interessanter.

Zwei Anregungen für verregnete Ferientage

```
      MAX
+   BOLTE                MAXUND
+   BUSCH              + MORITZ
---------              --------
= MORITZ               = WBUSCH
```

1. Zu jedem der beiden Zahlenrätsel existiert mindestens eine Lösung.
2. Man wähle ein geeignetes Zahlsystem.
3. Vor evtl. Abänderung von P17.4 schätze man die zu erwartende Laufzeit ab.
4. Der Kreativität sind beim Erfinden geeigneter Algorithmen keine Grenzen gesetzt!

18
Töne und Geräusche

Musik läuft im allgemeinen nicht so schnell ab, als daß man sie nicht auch in BASIC definieren könnte. Wir wollen aber in Abschnitt 19.4 auch noch Bewegung dazu darstellen, und dafür wäre BASIC viel zu langsam. Außerdem kann man von der Maschinensprache her den Tongenerator im 8253 sehr gut manipulieren.

18.1 Musik mit vorgegebenen Tönen

Zunächst betrachten wir die einem Evergreen nachempfundene Melodie:
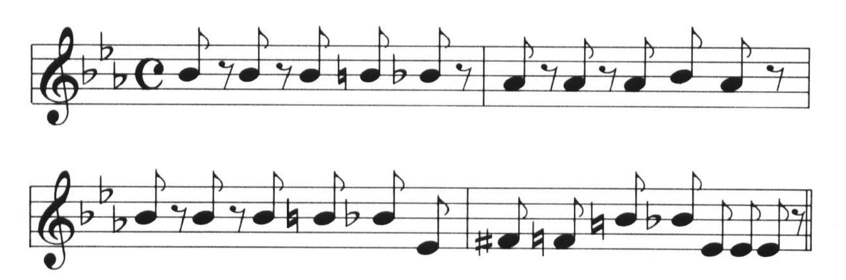


Bild 18.1 Die Melodie

Ihre Codierung ist auf den Seiten 63 ff. des Sharp-Handbuchs beschrieben. Zum Verständnis sollte man wissen, daß abweichend von unserer deutschen Bezeichnung im Amerikanischen (und von den Japanern übernommen) «unser Ton H» als «B» bezeichnet wird. Deshalb müssen wir «unser B» als «Ais», geschrieben «+A» ansprechen.

Obige Melodie wäre in BASIC wie folgt zu codieren:

+A3R+AR+AB+AR+GR+GR+G+A+GR+AR+AR+AB+A+D+FF
B+A+D+D+DR⟨CR⟩

Dabei steht 3 für Achtelnote und R für Pause (repose).

Für ein Maschinenprogramm wird das in ASCII-Werte umgesetzt und in unseren Datenbereich eingespeichert.

```
P18.1.1   (Prüfsumme: 0BC3)

1300 23 41 33 52 23 41 52 23 41 42 23 41
130C 52 23 47 52 23 47 52 23 47 23 41 23
1318 47 52 23 41 52 23 41 52 23 41 42 23
1324 41 23 44 23 46 46 42 23 41 23 44 23
1330 44 23 44 52 0D
```

Wir übergeben die Anfangsadresse dieses Strings im DE-Register und rufen die Monitorroutine MELDY auf. Zweckmäßigerweise verläßt man sich nicht darauf, daß mehr oder weniger per Zufall ein brauchbares Tempo vorgegeben ist, sondern bestimmt mittels XTEMP:

```
P18.1.2
2000  LD   A,04        3E 04      ;variieren
2002  CALL XTEMP       CD 41 00
2005  LD   DE,1300     11 00 13
2008  CALL MELDY       CD 30 00
200B  RST  00          C7
```

18.2 Manipulation der Tonerzeugung

Der Komfort fertig vorgegebener Töne ist nicht immer willkommen. Für den Maschinenprogrammierer ist es oft interessanter, näher an der Hardware manipulieren zu können. Die letzte Konsequenz werden wir nicht erreichen, denn unser MZ-700 ist mit einem Intervall-Timer 8253 ausgestattet, der u. a. die Uhr taktet, hier aber je nach eingegebenem Zählwert die diesem entsprechende Frequenz ausgibt.

Die Routine MELDY wird mit Adresse 0030 aufgerufen. Dort finden wir im Assemblerlisting C3 C7 01 bzw. JP 01C7 mit der symbolischen Adresse ?MELDY. Ab 01C7 werden zunächst drei Doppelregister gerettet und in 01D1 (DE) nach A geladen; das erste (oder später ein anderes)

Zeichen unseres Notenstrings steht jetzt im Akku. Wir nehmen an, daß das keins der Sonderzeichen –, +, D7, # ist, so daß wegen «NZ» von 01EF nach 01EF + 6 = 01F5 gesprungen wird. (Beachten Sie die von unserer Darstellungs- und Zählweise verschiedene Codierung des Sprungs!) An der neuen Stelle wird ONPU aufgerufen, ein Programmteil, der aus den Notenwerttabellen 026C...029A die vierstellige Hexadezimalzahl herausliest, die dem Intervall-Timer zu übergeben ist. Wurde die richtige Zahl gefunden, erfolgt von 0221 aus ein Sprung nach 022C (ONP2). Das E-Register wird aus der gefundenen richtigen Adresse (in HL) mit dem LSB des Delays (Zählverzögerung) geladen und nach INC HL das D-Register mit dem MSB. Nach einer längeren Verarbeitung trifft der Prozessor in 026B auf ein RET, das ihn nach 01FB zurückführt. Drei Zeilen tiefer finden wir CALL MLDST mit dem Kommentar MELODY START. Hilft das weiter? Wir suchen ab 02AB weiter. Zeile 02B4 lautet im Disassemblerformat LD HL,E004, und in den beiden nächsten Zeilen wird der Inhalt von DE (also das Delay) dem 8253 übergeben. Vergleichen Sie dazu die Aufstellung auf Seite 130 des Sharp-Handbuchs. Das regt uns zu einem Versuch an: Wir laden zunächst die Adresse von C0 des 8253 nach HL und den Phantasiewert 1111 nach DE, um dann mit CALL 02B7 mitten in MLDST hineinzuspringen. Beim weiteren Ausbau dieses Ansatzes stoßen wir auf Schwierigkeiten. Haben wir etwas übersehen? Zunächst funktioniert der Rücksprung nicht sicher, denn da folgt ja noch ein POP DE, das den Stapel verändert. Wir müßten also schon beim PUSH DE in 02B2 einspringen. Fehlt dann aber nicht die 8253-Adresse? Sehen Sie bitte in 02B4 genau nach! Dort wird sie ja unter dem symbolischen Namen CONTO nach HL geladen. Noch eins müssen wir beachten: In 02B3 werden die Inhalte von DE und HL ausgetauscht. Wir müssen das Delay also jetzt nach HL laden. Damit ergibt sich ein neuer Ansatz:

```
P18.2.1
3000  LD    HL,1111        21 11 11
3003  CALL  02B2           CD B2 02
3006  JR    3006           18 FE
```

Die Endlosschleife soll den bei Tonexperimenten störenden Meldeton des Monitors unterdrücken. Ein Dauerton ist die Folge. Wir hätten die Tonerzeugung mit MLDSP beenden müssen. Also wandeln wir den Schluß so ab:

```
3006  CALL MLDSP          CD BE 02
3009  HALT                76
```

Jetzt hören wir gar nichts; wir haben ja den Ton sofort nach Erzeugung unterdrückt. Es muß noch eine Verzögerung eingefügt werden:

```
P18.2.2
3006  LD   A,40         3E 40        ;variieren
3008  CALL WARTEN       CD B8 AF
300B  CALL MLDSP        CD BE 02
300E  CALL PAUSKY       CD D9 AF
3011  RST  00           C7
```

Wenn Sie nicht vergessen haben, das EBS zu laden, verläuft jetzt alles nach Wunsch, und wir versuchen eine Weiterentwicklung.

18.3 Effekt-Sirene

Wenn wir das Delay kontinuierlich ändern, müßte auch ein sich kontinuierlich ändernder Ton die Folge sein. Wir erzielen das, indem wir HL dekrementieren. Da in MLDST das HL-Register verändert wird, müssen wir es auf den Stapel retten. Daraus ergibt sich schließlich

```
P18.3.1
4000  LD   HL,1111      21 11 11
4003  PUSH HL           E5
4004  CALL TONAB        CD B2 02
4007  POP  HL           E1
4008  DEC  HL           2B
4009  LD   A,H          7C
400A  AND  A            A7
400B  JR   NZ,4003      20 F6
400D  CALL MLDSP        CD BE 02
4010  CALL SBMONI       CD B3 AF
4013  JR   4000         18 EB
```

SBMONI erlaubt uns, die Sirene zu stoppen. – Versuchen Sie auch andere Delay-Werte, vor allem in 4002.

A18.3.2 Schreiben Sie ein Programm für eine Sirene mit abwechselnd ansteigendem und abfallendem Ton.

19
Schnelle Grafik

Von den vielen Grafiksymbolen, die unser MZ-700 bereithält, benutzen wir jetzt die Bildschirmcodes F0 ... FF (vgl. Sharp-Handbuch Seite 159), um die Darstellungsmöglichkeit auf $(2 \star 40) \star (2 \star 25) = 4000$ Elemente zu verfeinern. Es lohnt sich, sich mit der Systematik der Codierung vertraut zu machen:

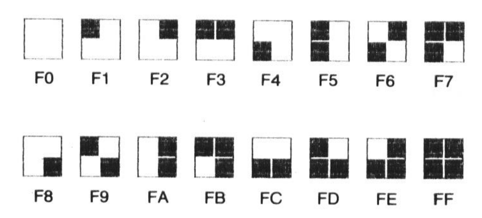


Bild 19.1 Die Viertelung der Schreibposition

Das MSN (most significant nibble) ist bei diesen Symbolen stets F. Das LSN ergibt sich als Summe der Werte der einzelnen Matrixviertel:

Links oben: 1	rechts oben: 2
links unten: 4	rechts unten: 8

Sollen alle vier Viertel gesetzt werden, errechnet sich der Bildschirmcode zu F0 + 1 + 2 + 4 + 8 = FF.

19.1 Bildschirmmanipulation mit Tabelle

Wir arbeiten nun mit folgender Tabelle:

```
P19.1.1   (Prüfsumme: 234C)

1300  D0 32 F0  D0 34 F1  D0 36 F2  D0 38 F3
130C  D0 82 F4  D0 84 F5  D0 86 F6  D0 88 F7
1318  D0 D2 F8  D0 D4 F9  D0 D6 FA  D0 D8 FB
1324  D1 22 FC  D1 24 FD  D1 26 FE  D1 28 FF
```

Die Tripel (Dreiergruppen) enthalten je ADDH (MSB der Bildschirmadresse), ADDL und Bildschirmcode. Die erste Eintragung bedeutet, daß in D032 des Bildschirmspeichers der Code F0 eingeschrieben werden soll. Wir sind dabei – ohne Nachteile im folgenden – um der besseren Lesbarkeit willen vom von Neumannschen Prinzip abgewichen.

Programmstruktur:

1. Bildschirm löschen.
2. B-Register für 16 Symbole mit 10h laden.
3. IX-Register auf Tabellenanfang setzen.
4. ADDH holen und H-Register damit laden.
5. ADDL holen und L-Register damit laden.
6. Bildschirmcode in den Akku holen.
7. ... und indirekt ins Video-RAM einspeichern.
8. IX dreimal inkrementieren.
9. B-Register dekrementieren und – wenn nicht null – zurück nach 4.
10. PAUSKY oder PAUSWK und Monitor.

A19.1.2 Schreiben Sie nach diesen Angaben ein Programm.

19.2 Drei Tanzpositionen

Wir arbeiten nun darauf hin, auf dem Bildschirm Bewegung vorzutäuschen, und stellen dazu einen Tänzer in drei Positionen dar. Die dazu benötigten Grafiksymbole speichern wir gleich zeilenweise um. Dazu setzen wir die Anfänge der Bildzeilen und die Position (Bild 19.2) in Tabellen um:

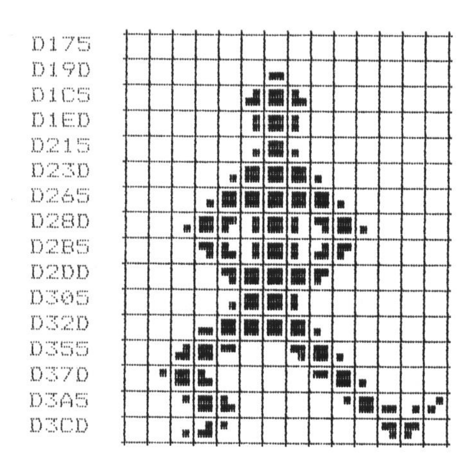


Bild 19.2 Tänzer, Position 1

Die Zahlen am linken Rand von Bild 19.2 geben die Bildzeilenanfänge an. Daraus folgt

```
P19.2.1   (Prüfsumme:   1532)
TBZANF   1400 D1 75 D1 9D D1 C5 D1 ED
         1408 D2 15 D2 3D D2 65 D2 8D
         1410 D2 B5 D2 DD D3 05 D3 2D
         1418 D3 55 D3 7D D3 A5 D3 CD
```

Nun muß Bild 19.2 in eine Tabelle der Bildschirmcodes umgesetzt werden:

```
P19.2.2 = TBPOS1   (Prüfsumme: D53E)

1420 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0
142E F0 F0 F0 F0 F0 F0 FC F0 F0 F0 F0 F0 F0 F0
143C F0 F0 F0 F0 F0 FE FF FD F0 F0 F0 F0 F0 F0
144A F0 F0 F0 F0 F0 FA FF F5 F0 F0 F0 F0 F0 F0
1458 F0 F0 F0 F0 F0 F8 FF F4 F0 F0 F0 F0 F0 F0
1466 F0 F0 F0 F0 F8 FF FF FF F4 F0 F0 F0 F0 F0
1474 F0 F0 F0 F8 FF FF FF FF FF F4 F0 F0 F0 F0
1482 F0 F0 F8 FF F7 FA FF F5 FB FF F4 F0 F0 F0
1490 F0 F0 F0 FB FD FA FF F5 FE F7 F0 F0 F0 F0
149E F0 F0 F0 F0 FB FF FF FF F7 F0 F0 F0 F0 F0
14AC F0 F0 F0 F0 F8 FF FF F5 F0 F0 F0 F0 F0 F0
```

```
14BA F0 F0 F0 FC FF FF FF FF F4 F0 F0 F0 F0 F0
14C8 F0 F0 FE FF F3 F0 F0 FB FF F4 F0 F0 F0 F0
14D6 F0 F2 FF FD F0 F0 F0 F0 F3 FF F4 F0 F0 F0
14E4 F0 F0 F2 FF FD F0 F0 F0 F0 F2 FF FC F8 F6
14F2 F0 F0 F8 FE F1 F0 F0 F0 F0 F0 F0 FB F7 F0
```

TBPOS1 enthält mehr Information als nötig. Viele F0s könnten eingespart werden. Wir wählen diesen Weg um der besseren Lesbarkeit willen und um die Programmgestaltung zu vereinfachen.

Im Hinblick auf die einheitliche Ausgabe aller drei Bilder verfassen wir folgendes Programm:

```
P19.2.3  (Prüfsumme: 132F)

START  2000  CALL CLS          CD 9F AF
       2003  JP   POS1         C3 29 20
BILD   2006  LD   B,10         06 10
       2008  LD   IX,TBZANF    DD 21 00 14
LAB0   200C  PUSH BC           C5
       200D  LD   H,(IX+00)    DD 66 00     ;H(hin)
       2010  LD   L,(IX+01)    DD 6E 01     ;L(hin)
       2013  LD   B,0E         06 0E        ;14 Symb.
LAB1   2015  LD   A,(IY+00)    FD 7E 00     ;IY=her
       2018  LD   (HL),A       77
       2019  INC  IY           FD 23
       201B  INC  HL           23
       201C  DJNZ LAB1         10 F7
       201E  INC  IX           DD 23
       2020  INC  IX           DD 23
       2022  POP  BC           C1
       2023  DJNZ LAB0         10 E7
       2025  NOP               00           ;später
       2026  NOP               00           ;hier
       2027  NOP               00           ;Musik
       2028  RET               C9
POS1   2029  LD   IY,TBPOS1    FD 21 20 14
       202D  CALL BILD         CD 06 20
       2030  HALT              76           ;(RET)
```

Wir testen das Programm in diesem Zustand, sind zufrieden und saven: S120020302000. Ganz genauso stellen wir die Tabelle für Position 2 auf:

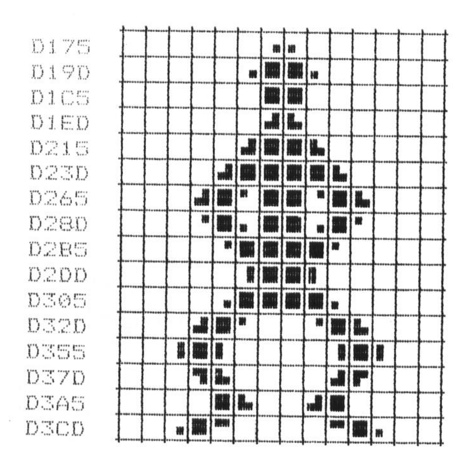


Bild 19.3 Tänzer, Position 2

```
P19.2.4 = TBPOS2   (Prüfsumme: D566)

1500 F0 F0 F0 F0 F0 F0 F8 F4 F0 F0 F0 F0 F0 F0
150E F0 F0 F0 F0 F0 F8 FF FF F4 F0 F0 F0 F0 F0
151C F0 F0 F0 F0 F0 F0 FF FF F0 F0 F0 F0 F0 F0
152A F0 F0 F0 F0 F0 F0 FE FD F0 F0 F0 F0 F0 F0
1538 F0 F0 F0 F0 F0 FE FF FF FD F0 F0 F0 F0 F0
1546 F0 F0 F0 F0 FE FF FF FF FF FD F0 F0 F0 F0
1554 F0 F0 F0 FE FF F1 FF FF F2 FF FD F0 F0 F0
1562 F0 F0 F0 F2 FF F4 FF FF F8 FF F1 F0 F0 F0
1570 F0 F0 F0 F0 F2 FF FF FF FF F1 F0 F0 F0 F0
157E F0 F0 F0 F0 F0 FA FF FF F5 F0 F0 F0 F0 F0
158C F0 F0 F0 F0 F8 FF FF FF FF F4 F0 F0 F0 F0
159A F0 F0 F0 F8 FF F7 F0 F0 FB FF F4 F0 F0 F0
15A8 F0 F0 F0 FF FF F0 F0 F0 F0 FF FF F0 F0 F0
15B6 F0 F0 F0 FA FF F0 F0 F0 F0 FF F5 F0 F0 F0
15C4 F0 F0 F0 F0 FF FD F0 F0 FE FF F0 F0 F0 F0
15D2 F0 F0 F8 FF F3 F0 F0 F0 F0 F3 FF F4 F0 F0
```

```
P19.2.5
POS2    2031   LD    IY,TBPOS2    FD 21 00 15
        2035   CALL  BILD         CD 06 20
        2038   HALT               76            ;(RET)
```

Testen Sie diesen Abschnitt und vergessen Sie (2004) nicht.

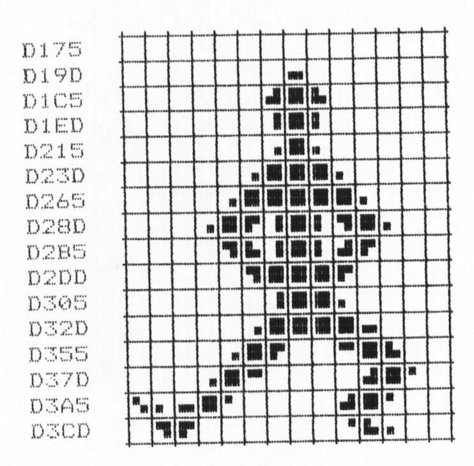


Bild 19.4 Tänzer, Position 3

A19.2.6 Stellen Sie TBPOS3 auf und dazu POS3.

Nach diesen Vorbereitungen starten wir den ersten großen Probelauf. Weil die Bilder hundertfach zu schnell gezeigt werden würden, benötigen wir eine Verzögerungsroutine und nehmen auch dazu das EBS zu Hilfe:

```
P19.2.7
PAUSE  2041  LD   A,20        3E 20       ;var.
       2043  CALL WARTEN      CD B8 AF
       2046  RET              C9
TEST   2047  CALL POS1        CD 29 20
       204A  CALL POS2        CD 31 20
       204D  CALL POS3        CD 39 20
       2050  CALL POS2        CD 31 20
       2053  JR   TEST        18 F2
```

Damit alles korrekt abläuft, sind folgende Änderungen nötig:

```
       2003  JP   TEST        C3 47 20
       2025  CALL PAUSE       CD 41 20
```

```
        2030   RET                 C9
        2038   RET                 C9
        2040   RET                 C9
Prüfsumme (2000 ... 2054) = 2469
```

19.3 Tanz und Musik

Wir fügen jetzt noch Bild und Ton zusammen. Teilaufgaben:
1. Notentabelle bereitstellen.
2. Ablauftempo mit XTEMP bestimmen. Dadurch wird PAUSE überflüssig.
3. Bild und Ton nach unserer Melodie koordinieren.

```
P19.3.1   (Prüfsumme 045F)
TABTON 16C0   23 44 33 0D 46 33 0D 0D 23 46 33 0D
       16CC   23 47 33 0D 23 41 33 0D 42 33 0D 0D
       16D8   52 33 0D 0D
```

```
P19.3.2   (Prüfsummen am rechten Rand)
START   2000   CALL  CLS          CD 9F AF
        2003   JP    TNZMUS       C3 41 20
        ....   ....  ......       .. .. ..
        2025   CALL  MELDY        CD 30 00
        ....   ....  ......       .. .. ..
TNZMUS  2041   LD    A,04         3E 04
        2043   CALL  XTEMP        CD 41 00        ;1CA5
LAB2    2046   LD    DE,TON+A     11 D0 16
        2049   CALL  POS1         CD 29 20
        204C   LD    DE,TONR      11 D8 16
        204F   CALL  POS2         CD 31 20
        2052   LD    DE,TON+A     11 D0 16
        2055   CALL  POS3         CD 39 20
        2058   LD    DE,TONR      11 D8 16
        205B   CALL  POS2         CD 31 20
        205E   LD    DE,TONA      11 D0 16
        2061   CALL  POS1         CD 29 20
        2064   LD    DE,TONB      11 D4 16
        2067   CALL  POS2         CD 31 20
        206A   LD    DE,TON+A     11 D0 16
        206D   CALL  POS3         CD 39 20
        2070   LD    DE,TONR      11 D8 16
```

```
2073  CALL POS2       CD 31 20
2076  LD   DE,TON+G   11 CC 16
2079  CALL POS1       CD 29 20
207C  LD   DE,TONR    11 D8 16       ;305B
207F  CALL POS2       CD 31 20
2082  LD   DE,TON+G   11 CC 16
2085  CALL POS3       CD 39 20
2088  LD   DE,TONR    11 D8 16
208B  CALL POS2       CD 31 20
208E  LD   DE,TON+G   11 CC 16
2091  CALL POS1       CD 29 20
2094  LD   DE,TON+A   11 D0 16
2097  CALL POS2       CD 31 20
209A  LD   DE,TON+G   11 CC 16
209D  CALL POS3       CD 39 20
20A0  LD   DE,TONR    11 D8 16
20A3  CALL POS2       CD 31 20
20A6  JR   LAB2       18 9E          ;3ECF
```

Es sei dem Leser überlassen, das Programm weiter zu vervollständigen.

19.4 Wir relokatieren Tanz und Musik

Abgesehen von einer eventuellen Vervollständigung weist P19.3. zwei Mängel auf:

1. Es läßt zwischen 16DB und 2000 fast 3 KB Speicherraum unbenutzt. Das ist nicht schlimm, denn Speicherraum haben wir genug. Unangenehmer ist schon, daß man zum Saven mehr Band benötigt und deswegen das Laden unnötig lange dauert.
2. Es werden nur 6 Bytes des EBS benutzt; dafür sollte man nicht zweimal laden müssen.

Zu 2: Wir ändern den Programmanfang ab zu

```
START 1FFE   LD     A,16        3E 16
      2000   CALL   PRNT        CD 12 00
```

A19.4.1 a) Schieben Sie das Programm an seine Daten heran, 1FFE...20A7 nach 16DC...
b) Erledigen Sie auch die innere Anpassung!

20
Bildschirm und Farbe

Das Video-RAM des MZ-700 reicht von D000 bis DFFF, letztlich unterteilt in vier Bereiche:

D000 ... D3E7	ist sichtbar und enthält eingespeicherte Bildschirmcodes;			
D3E8 ... D7FF	unsichtbar, Bildschirmcodes;			
D800 ... DBE7	Farbinformation für sichtbaren Bereich;			
DBE8 ... DFFF	Farbinformation für unsichtbaren Bereich.			
Codierung:	Schwarz: 0		Grün:	4
	Blau: 1		Hellblau:	5
	Rot: 2		Gelb:	6
	Purpur: 3		Weiß:	7

Die 64 möglichen zweistelligen Farbcodierungen (der zweite Zeichensatz bleibt unberührt) setzen sich aus diesen acht Ziffern zusammen; das MSN bestimmt die Farbe des dargestellten Zeichens, das LSN die Hintergrundfarbe.

20.1 Farbgebung – Zufallsfarben

Der normalen Farbgebung entspricht der Farbcode 71; Sie können das mit DD800 sehr bequem nachprüfen. Sie können aber auch mit MD800 (oder MD850) andere Farbcodes eingeben. Sie sehen dann sofort die Auswirkung – sofern Sie einen Farbmonitor oder einen Farbfernseher besitzen. Das regt zum Improvisieren an: Auf dem Bildschirm sollen von Zufallszahlen gesteuerte Farbmuster entstehen. Grobaufbau:

1. Uhr stellen mit TIMST.
2. L aus E005 laden.
3. Kurze Verzögerung.

4. E005 auslesen, auf Werte D8...DF maskieren und nach H laden.
5. Kurze Verzögerung.
6. E005 auslesen und auf 77 begrenzen.
7. Indirekt in Farb-RAM speichern.
8. Kurze Verzögerung.
9. Rücksprung nach 2.

```
P20.1.1  (Prüfsumme: 11D5)

RNDCOL 2000  XOR  A          AF           ;1.
       2001  LD   DE,1111    11 11 11
       2004  CALL TIMST      CD 33 00
LABEL  2007  LD   A,(E005)   3A 05 E0     ;2.
       200A  LD   L,A        6F
       200B  LD   B,FF       06 FF        ;3.
L1     200D  DJNZ L1         10 FE
       200F  LD   A,(E005)   3A 05 E0     ;4.
       2012  AND  07         E6 07
       2014  OR   D8         F6 D8
       2016  LD   H,A        67
       2017  LD   B,FF       06 FF        ;5.
L2     2019  DJNZ L2         10 FE
       201B  LD   A,(E005)   3A 05 E0     ;6.
       201E  AND  77         E6 77
       2020  LD   (HL),A     77           ;7.
       2021  LD   B,FF       06 FF        ;8.
L3     2023  DJNZ L3         10 FE
       2025  JR   LABEL      18 E0
```

Ersetzen Sie versuchsweise in Zeile 200B LD B,FF durch LD B,A und NOP. Veränderung? Begründung?

20.2 Simulation einer Verkehrsampel

Wir wollen jetzt auf dem Bildschirm eine Verkehrsampel in der richtigen Reihenfolge rot, rot/gelb, grün, gelb, rot usw. blinken lassen. Wir löschen dazu den Bildschirm und erzeugen als Untergrund bzw. Rahmen auf dem blauen Schirm ein 4 ⋆ 7 Felder großes Rechteck, in dem die drei Lampen untergebracht werden sollen. Erster Schritt:

```
P20.2.1
INIT     3000   CALL  CLS          CD 9F AF
         3003   LD    HL,D826      21 62 D8 ;li ob Ecke
         3006   LD    (HL),00      36 00
         3008   INC   HL           23
         3009   LD    (HL),00      36 00
         300B   INC   HL           23
         300C   LD    (HL),00      36 00
         300E   HALT               76
```

Hiermit erhalten wir drei nebeneinanderliegende schwarze Felder. Wir benötigen das sechs weitere Male darunter und erzielen dies, indem wir einen geeigneten Speicherbereich sechsmal über sich selbst hinweg kopieren.:

```
P20.2.2
         300E   LD    B,06         06 06
LAB0     3010   PUSH  BC           C5
         3011   LD    HL,D858      21 58 D8
         3014   LD    DE,D880      11 80 D8
         3017   LD    BC,0118      01 18 01 ;7*40d=118h
         301A   LDIR               ED B0
         301C   POP   BC           C1
         301D   DJNZ  LAB0         10 F1
         301F   HALT               76
```

Sodann benötigen wir eine Routine, die alle drei Lampen löscht, damit wir nicht einmal diese und einmal jene zu löschen brauchen:

```
P20.2.3
AUS      3022   XOR   A            AF       ;schwarz
         3023   LD    (D88B),A     32 8B D8 ;rt Lampe
         3026   LD    (D8DB),A     32 DB D8 ;ge Lampe
         3029   LD    (D92B),A     32 2B D9 ;gr Lampe
         302C   RET                C9
```

Nun müssen wir uns Gedanken über die Laufzeit machen. WARTEN liefert etwa folgende Verzögerungen:

A =	50	60	70	80	90	A0	B0	C0	D0	E0	F0
	1,5	2	3	4	5	7	10	13	16	20	25 s

Für unsere Ampelsimulation wählen wir folgende Zeiten:

Rot:	9 s	entspr.	A = AB
Rot/Gelb:	1,5 s	entspr.	A = 50
Grün:	6 s	entspr.	A = 98
Gelb:	3 s	entspr.	A = 70

Damit ergeben sich folgende Farbroutinen:

```
P20.2.4
ROT     302D  LD    A,22          3E 22
        302F  LD    (D88B),A      32 8B D8
        3032  LD    A,AB          3E AB
        3034  CALL  WARTEN        CD B8 AF
        3037  RET                 C9
ROTG    3038  LD    A,22          3E 22
        303A  LD    (D88B),A      32 8B D8
        303D  LD    A,66          3E 66
        303F  LD    (D8DB),A      32 DB D8
        3042  LD    A,50          3E 50
        3044  CALL  WARTEN        CD B8 AF
        3047  RET                 C9
GRUE    3048  LD    A,44          3E 44
        304A  LD    (D92B),A      32 2B D9
        304D  LD    A,90          3E 90
        304F  CALL  WARTEN        CD B8 AF
        3052  RET                 C9
GELB    3053  LD    A,66          3E 66
        3055  LD    (D8DB),A      32 DB D8
        3058  LD    A,70          3E 70
        305A  CALL  WARTEN        CD B8 AF
        305D  RET                 C9
```

Danach fehlt nur noch das Hauptprogramm:

```
P20.2.5
AMPEL   305E  CALL  AUS           CD 22 30
        3061  CALL  ROT           CD 2D 30
        3064  CALL  AUS           CD 22 30
        3067  CALL  ROTG          CD 38 30
        306A  CALL  AUS           CD 22 30
        306D  CALL  GRUE          CD 48 30
        3070  CALL  AUS           CD 22 30
        3073  CALL  GELB          CD 53 30
        3076  JR    AMPEL         18 E6
```

```
Und es ist einzufügen

        301F   JR    AMPEL          18 3D
        3021   NOP                  00

Prüfsumme (3000 ... 3077) = 35E8
```

Das Programm weist noch drei Mängel auf:

1. Nicht alle AUS-Aufrufe sind nötig.
2. Jede Farbroutine endet mit WARTEN, RET. Ließe sich das sparsamer einbauen?
3. Könnte man WARTEN in die AUS-Routine einbauen?
4. Stimmt der schwarze Rahmen der Ampel?

Es sei dem Leser überlassen, Vereinfachungen und Korrekturen anzubringen.

Eine Ampel steht selten allein. Das Problem wird etwas komplizierter, wenn man auch die «unsere Straße» kreuzende Querstraße mit einbezieht und beide Ampeln möglichst differenziert aufeinander abstimmt. Daher ergeht an den Leser die Anregung, nebeneinander zwei korrespondierende Ampeln auf dem Bildschirm erscheinen zu lassen.

```
Bild (fast) ohne Worte

rot    ¯¯¯¯¯l__________l¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯l__________l¯¯¯¯

gelb   ___l¯l______l¯¯l____________l¯l______l¯¯l____

grün   _____l¯¯¯¯¯¯¯l_________________l¯¯¯¯¯¯¯l______

rot    __l¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯l__________l¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯l__

gelb   ¯¯l____________l¯l______l¯¯l____________l¯l__

grün   ___________________l¯¯¯¯¯¯¯l________________l¯¯
```

21
Drucken mit Maschinenprogramm

Da nicht alle Leser über einen Drucker verfügen werden, wollen wir diesen Abschnitt kurz halten. Zudem muß die Einschränkung gemacht werden, daß der Verfasser mit seinem MZ-700 einen EPSON MX 80 über ein Voigt-Interface betreibt. Dennoch darf davon ausgegangen werden, daß das Gesagte auch für den Printerplotter des MZ-731 gilt.

21.1 Die Monitorroutine PMSG

Wie das Sharp-Handbuch auf Seite 149 beschreibt, genügt zum Ansprechen des Druckers bereits der P-Befehl: Gibt man z. B. "PMUEN CHEN" ein, druckt der Printer «MUENCHEN» aus.

Wir betreiben nun wieder Fährtensuche im Assemblerlisting des Monitors: Bei 00D5 finden wir den Buchstaben P. Von dort wird mit der Sprungweite 7C nach 0155 verwiesen. Wir nehmen an, daß dem P kein & folgt. Dann geht es bei PTST1 in 0170 weiter. Dort wird PMSG in 01A5 aufgerufen, was doch wohl printer message bedeutet. Wir finden auf page 05 in Zeile 59 den Hinweis, daß die Datenadresse in DE steht und die Botschaft mit 0D endet. Das vollziehen wir nach:

```
P21.1.1
1400   4D 5A 2D 37 30 30 20 4F 4B 0D
4000   LD   DE,1400        11 00 14
4003   CALL PMSG           CD A5 01
4006   RST  00             C7
```

Eventuell muß in die Botschaft noch 10, der Code für «*line feed*» eingefügt werden. Wenn Sie nicht vergessen haben, Ihren Drucker einzuschalten, müßte eigentlich alles nach Wunsch verlaufen. Das regt uns zu einer «ernsthaften» Anwendung an.

21.2 Wir drucken einen Speicherauszug

Zunächst eine Vorübung. Von 0124 an sollen mit Adresse am linken Rand 16 Bytes ausgegeben werden. (Je nach Drucker müßte diese Anzahl evtl. begrenzt werden.) Zum Aufbau der codierten Botschaft wählen wir in unserem Arbeitsbereich eine Stelle, die wir kaum mit anderen Daten belegen werden: 1300... Als Pointer hierin benutzen wir das IX-Register, zum Holen der Daten HL.

Die Subroutine SPC fügt jeweils ein codiertes space in den aufzubauenden String ein. Ihren Hauptteil ab 2004 benutzen wir auch zum Anreihen der Datencodes.

Die Subroutine UMWSP wandelt die Daten aus dem Hexadezimalformat ins ASCII-Format um und fügt sie an.

Im Hauptprogramm PDUMP2 geschieht folgendes:

1. Zuerst werden HL und IX auf ihre Anfangswerte gesetzt.
2. ADDH und ADDL werden umgewandelt und angefügt.
3. Es folgt ein space.
4. B wird auf 10h gesetzt für 16 Speicherstellen.
5. (HL) wird geholt, umgewandelt und angefügt. Es folgt ein space, und HL wird um 1 erhöht.
6. Wenn noch nicht 16mal gelesen, zurück nach 5.
7. Ein ⟨CR⟩ wird angehängt. Benötigt Ihr Drucker noch ein ⟨LF⟩ (10h)?
8. Aufruf von PMSG und restart.

```
P21.2.1  (Prüfsumme: 1B6A)

PDUMPV 2000  JR    PDUMP2       18 1D
SPC    2002  LD    A,20         3E 20
SPC2   2004  LD    (IX+00),A    DD 77 00
       2007  INC   IX           DD 23
       2009  RET                C9
UMWSP  200A  PUSH  AF           F5
       200B  SRL   A            CB 3F
       200D  SRL   A            CB 3F
       200F  SRL   A            CB 3F
       2011  SRL   A            CB 3F
       2013  CALL  ASC          CD DA 03
       2016  CALL  SPC2         CD 04 20
       2019  POP   AF           F1
       201A  CALL  ASC          CD DA 03
       201D  JR    SPC2         18 E5
```

```
PDUMP2 201F  LD    HL,0124       21 24 01
       2022  LD    IX,1300       DD 21 00 13
       2026  LD    A,H           7C
       2027  CALL  UMWSP         CD 0A 20
       202A  LD    A,L           7D
       202B  CALL  UMWSP         CD 0A 20
       202E  CALL  SPC           CD 02 20
       2031  LD    B,10          06 10
LOOP   2033  LD    A,(HL)        7E
       2034  CALL  UMWSP         CD 0A 20
       2037  CALL  SPC           CD 02 20
       203A  INC   HL            23
       203B  DJNZ  LOOP          10 F6
       203D  LD    A,0D          3E 0D
       203F  CALL  SPC2          CD 04 20
       2042  LD    DE,1300       11 00 13
       2045  CALL  PMSG          CD A5 01
       2048  RST   00            C7
```

Nach diesem erfolgreichen Vortest gehen wir nun an die Gestaltung des vollständigen Programms, wobei wir 2000...201E übernehmen. Anfangs- und Endadresse des gewünschten Hexdumps und die Anzahl der Daten je Zeile übergeben wir dem Programm in unserem Arbeitsbereich 1200...1204:

1200 BL BH VL VH PZ

Darin bedeuten

VH:	Von-Adresse MSB	VL:	Von-Adresse LSB
BH:	Bis-Adresse MSB	BL:	Bis-Adresse LSB
PZ:	Daten je Zeile		

Grobstruktur:

1. Wir holen mit Hilfe von IX die Von-Adresse nach HL, die Bis-Adresse nach DE und die Anzahl der Bytes je Zeile nach 2045.
2. Den Zeilenanfang gestalten wir wie in P21.2.1
3. Den Aufbau der eigentlichen Druckzeile verlegen wir in eine Schleife, die mit LD B,PZ begrenzt wird (Zeile 026).
4. Nachdem ein Byte geholt, umgewandelt und angefügt wurde,
5. werden HL und DE auf den Stapel gerettet, vertauscht und subtrahiert.

6. Wurde die Differenz negativ, so haben wir das abgesteckte Intervall überschritten und verlassen die Schleife.
7. Anderenfalls werden DE und HL wieder auf die benötigten Zwischenwerte gesetzt und die Schleife fortgesetzt, wenn die Zeile noch nicht voll ist.
8. Am Ende der Zeile wird ⟨CR⟩ angehängt und DE gerettet.
9. DE zeigt auf Anfang Botschaft. Drucken.
10. DE wiederherstellen und zurück zu neuer Zeile.
11. Zum Abschluß ⟨CR⟩ anfügen und drucken. Restart. Auf den schlampigen Schluß wird ausdrücklich aufmerksam gemacht: In 2029 und 202A werden HL und DE gepushet. Wir verzichten auf eine korrekte Restaurierung des Stapels, weil durch RST 00 das System ohnehin neu initialisiert wird.

Es versteht sich von selbst, daß die neuen Programme und Routinen nur dann richtig ablaufen, wenn das jeweils bisherige EBS geladen ist.

```
P21.2.2   (Prüfsumme:  54B0)

015 2000 11 92 20     PDUMP:   LD   DE,TPDUMP
016 2003 CD ED AF              CALL MSGNL
017 2006 CD 46 20              CALL PDEIN
018 2009 2A 02 12     SETREG:  LD   HL,(VON)
019 200C ED 5B 00 12           LD   DE,(BIS)
020 2010 DD 21 00 13  LZEILE:  LD   IX,1300H
021 2014 7C                    LD   A,H
022 2015 CD 68 20              CALL UMWANH
023 2018 7D                    LD   A,L
024 2019 CD 68 20              CALL UMWANH
025 201C CD 8A 20              CALL SPCANH
026 201F 06 10        LB/Z:    LD   B,10H
027 2021 7E           BYTANH:  LD   A,(HL)
028 2022 CD 68 20              CALL UMWANH
029 2025 CD 8A 20              CALL SPCANH
030 2028 23                    INC  HL
031 2029 E5                    PUSH HL
032 202A D5                    PUSH DE
033 202B A7                    AND  A
034 202C EB                    EX   DE,HL
035 202D ED 52                 SBC HL,DE
036 202F FA 42 20              JP   M,PDENDE
037 2032 D1                    POP  DE
038 2033 E1                    POP  HL
```

```
039 2034 10 EB                   DJNZ BYTANH
040 2036 3E 0D                   LD   A,0DH
041 2038 CD 8C 20                CALL SPC2
042 203B D5                      PUSH DE
043 203C CD 7E 20                CALL ENDEZL
044 203F D1                      POP  DE
045 2040 18 CE                   JR   LZEILE
046 2042                ;
047 2042 CD 7E 20       PDENDE:  CALL ENDEZL
048 2045 C7                      RST  00
049 2046                ;
050 2046 11 29 AE       PDEIN:   LD   DE,TVON
051 2049 CD 15 00                CALL MSG
052 204C 21 03 12                LD   HL,VON+01
053 204F CD 4C AF                CALL IN4HEX
054 2052 11 2E AE                LD   DE,TBIS
055 2055 CD 15 00                CALL MSG
056 2058 CD 4C AF                CALL IN4HEX
057 205B 11 AA 20                LD   DE,TBYT/Z
058 205E CD 15 00                CALL MSG
059 2061 CD 57 AF                CALL INBYTE
060 2064 32 20 20                LD   (LB/Z+01),A
061 2067 C9                      RET
062 2068                ;
063 2068 F5             UMWANH:  PUSH AF
064 2069 CB 3F                   SRL  A
065 206B CB 3F                   SRL  A
066 206D CB 3F                   SRL  A
067 206F CB 3F                   SRL  A
068 2071 CD DA 03                CALL ASC
069 2074 CD 8C 20                CALL SPC2
070 2077 F1                      POP  AF
071 2078 CD DA 03                CALL ASC
072 207B C3 8C 20                JP   SPC2
073 207E                ;
074 207E 3E 0D          ENDEZL:  LD   A,0DH
075 2080 CD 8C 20                CALL SPC2
076 2083 11 00 13                LD   DE,1300H
077 2086 CD A5 01                CALL PMSG
078 2089 C9                      RET
079 208A                ;
080 208A 3E 20          SPCANH:  LD   A,20H
081 208C DD 77 00                LD   (IX+00),A
082 208F DD 23                   INC  IX
083 2091 C9                      RET
084 2092                ;
```

```
085 2092 16          TPDUMP: DEFB 16H
086 2093 53 9E 92 A6         DEFM 'Speicherauszug'
087 2097 9F 98 92 9D
088 209B A1 A5 A4 A2
089 209F A5 97
090 20A1 20 9C 9D A5         DEFM ' drucken'
091 20A5 9F A9 92 B0        \
092 20A9 0D                  DEFB 0DH
093 20AA 20 42 BD 96 TBYT/Z: DEFM ' Bytes je Zeile'
094 20AE 92 A4 20 AF
095 20B2 92 20 5A 92
096 20B6 A6 B8 92 20
097 20BA 0D                  DEFB 0DH
098 20BB             ;
099 20BB                     END
```

22
Vier Utilities fürs EBS

22.1 VERSCH verschiebt Speicherblöcke

P21.2.2 ist sehr nützlich und soll ins EBS aufgenommen werden. Ein Umschreiben von Hand ist bei 186 Bytes kein Vergnügen, und für jede Verschiebung einen Blocktransfer hinschreiben zu müssen, ist auf die Dauer auch nicht anregend. Deshalb konzipieren wir eine Routine, die das ein für allemal erledigt. Wir legen dazu im Arbeitsbereich folgende Adressen fest:

1200 NL NH BL BH VL VH

Das ist so zu lesen: Der zu verschiebende Block reicht VON (VH,VL) BIS (BH,BL), und sein Anfang soll NACH (NH,NL) verschoben werden.

Zu einzelnen Programmabschnitten:
ANZAHL ermittelt die Anzahl der zu verschiebenden Bytes durch Subtraktion BIS – VON mit nachfolgendem INC HL.
V/R? prüft durch Subtraktion NACH – VON, ob vorwärts (in Programmablaufrichtung) oder rückwärts zu verschieben ist.

```
P22.1.1   (Prüfsumme: 311D)

15 3000 11 56 30     VERSCH: LD   DE,TVERS
16 3003 CD 15 00             CALL MSG
17 3006 CD 06 00     VERS2:  CALL LETNL
18 3009 11 29 AE             LD   DE,TVON
19 300C CD 15 00             CALL MSG
20 300F 21 05 12             LD   HL,VON+01
21 3012 CD 4C AF             CALL IN4HEX
22 3015 11 2E AE             LD   DE,TBIS
23 3018 CD 15 00             CALL MSG
24 301B CD 4C AF             CALL IN4HEX
25 301E 11 72 30             LD   DE,TNACH
```

```
26 3021 CD 15 00                    CALL  MSG
27 3024 CD 4C AF                    CALL  IN4HEX
28 3027 A7                 ANZAHL:  AND   A
29 3028 2A 02 12                    LD    HL,(BIS)
30 302B ED 5B 04 12                 LD    DE,(VON)
31 302F ED 52                       SBC   HL,DE
32 3031 23                          INC   HL
33 3032 E5                          PUSH  HL
34 3033 C1                          POP   BC
35 3034 A7                 V/R?:    AND   A
36 3035 2A 00 12                    LD    HL,(NACH)
37 3038 ED 52                       SBC   HL,DE
38 303A FA 4B 30                    JP    M,RUECK
39 303D 2A 00 12           VORW:    LD    HL,(NACH)
40 3040 09                          ADD   HL,BC
41 3041 2B                          DEC   HL
42 3042 E5                          PUSH  HL
43 3043 D1                          POP   DE
44 3044 2A 02 12                    LD    HL,(BIS)
45 3047 ED B8                       LDDR
46 3049 18 BB                       JR    VERS2
47 304B                    ;
48 304B 2A 04 12                    LD    HL,(VON)
49 304E ED 5B 00 12                 LD    DE,(NACH)
50 3052 ED B0                       LDIR
51 3054 18 B0                       JR    VERS2
52 3056                    ;
53 3056 16                 TVERS:   DEFB  16H
54 3057 53 9E 92 A6                 DEFM  'Speicherbl'
55 305B 9F 98 92 9D
56 305F 9A B8
57 3061 BA                          DEFB  BAH
58 3062 9F A9 92 20                 DEFM  'cke ver'
59 3066 AB 92 9D A4
60 306A 9F 98 A6 92
61 306E 9A 92 B0
62 3071 0D                          DEFB  0DH
63 3072 20 B0 A1 9F TNACH:          DEFM  ' nach '
64 3076 98 20
65 3078 0D                          DEFB  0DH
66 3079                    ;
67 3079                             END
```

A22.1.2 a) Schieben Sie PDUMP vor das bisherige EBS. a1) Innere Anpassung?
b) Machen Sie dasselbe mit VERSCH.

22.2 VERIFY zum Überprüfen von Speicherblöcken

Wir verfassen nun ein Programm, das es gestattet, Speicherinhalte – dazu gehören auch Programme – mit einigem Komfort mit der Vorlage zu vergleichen. Das kann vor allem bei der Kontrolle neu eingetippter längerer Programme hilfreich sein.

Zuerst wird die Adresse eingegeben, von der an man kontrollieren will, dann der Prüfsummenstand, mit dem man beginnen will; wird im allgemeinen 0000 sein.

VERIFY gibt dann die Adresse der momentan angesprochenen Speicherstelle und deren Inhalt aus. Man hat nun drei Antwortmöglichkeiten:
⟨CR⟩ bedeutet, daß der angezeigte Speicherinhalt akzeptiert wird. Danach Ausgabe der Prüfsumme bis zu dieser Speicherstelle und neue Zeile.
"K" bedeutet «Korrektur». Man kann nun den gewünschten neuen Inhalt eingeben. Danach weiter wie bei CR.
"R" kann man eingeben, wenn man in der Kontrolle um (je) einen Schritt zurückgreifen möchte.

Wir verlegen das Programm gleich an seinen endgültigen Platz.

```
P22.2.1   (Prüfsummen am rechten Rand)

22 ABD7 1162AC    VERIFY: LD    DE,TVERFY        ;011F
23 ABDA CD1500            CALL  MSG
24 ABDD 210312            LD    HL,VON+01
25 ABE0 CD4CAF            CALL  IN4HEX
26 ABE3 CD0600            CALL  LETNL            ;04D2
27 ABE6 1170AC            LD    DE,TPRF
28 ABE9 CD1500            CALL  MSG
29 ABEC CD4CAF            CALL  IN4HEX
30 ABEF CD0600    LOOP:   CALL  LETNL
31 ABF2 CDF7AB            CALL  ZEILBG           ;0BEB
32 ABF5 18F8              JR    LOOP
33 ABF7 210312    ZEILBG: LD    HL,VON+01
34 ABFA CD25AF            CALL  OUT4HX
35 ABFD CD0C00            CALL  PRINTS
36 AC00 2A0212            LD    HL,(VON)         ;0FE9
37 AC03 7E                LD    A,(HL)
38 AC04 CD30AF            CALL  OUTBYT
39 AC07 CD0C00            CALL  PRINTS
```

```
40 AC0A CDB309   LZEIL0: CALL ??KEY
41 AC0D F5               PUSH AF          ;156A
42 AC0E CDCE0B           CALL ?DACN
43 AC11 CD1200           CALL PRNT
44 AC14 CD0C00           CALL PRINTS
45 AC17 F1               POP  AF          ;19B9
46 AC18 FECD             CP   CDH         ;<CR>
47 AC1A 280A             JR   Z,VRFWTR
48 AC1C FE12             CP   12H         ;'R'
49 AC1E 2824             JR   Z,VRFRCK
50 AC20 FE0B             CP   0BH         ;'K'
51 AC22 2835             JR   Z,VFKORR    ;1E78
52 AC24 18E4             JR   LZEIL0
53 AC26 5E       VRFWTR: LD   E,(HL)
54 AC27 1600             LD   D,00
55 AC29 DD2A0012         LD   IX,(SUMME)
56 AC2D DD19             ADD  IX,DE       ;21F7
57 AC2F DD220012         LD   (SUMME),IX
58 AC33 23               INC  HL
59 AC34 220212           LD   (VON),HL
60 AC37 3A0112           LD   A,(SUMME+01)
61 AC3A CD30AF           CALL OUTBYT      ;255A
62 AC3D 3A0012           LD   A,(SUMME)
63 AC40 CD30AF           CALL OUTBYT
64 AC43 C9               RET
65 AC44 2B       VRFRCK: DEC  HL
66 AC45 220212           LD   (VON),HL    ;287C
67 AC48 5E               LD   E,(HL)
68 AC49 1600             LD   D,00
69 AC4B 2A0012           LD   HL,(SUMME)
70 AC4E A7               AND  A
71 AC4F ED52             SBC  HL,DE       ;2B12
72 AC51 220012           LD   (SUMME),HL
73 AC54 CD0600           CALL LETNL
74 AC57 189E             JR   ZEILBG
75 AC59 CD57AF   VFKORR: CALL INBYTE
76 AC5C 77               LD   (HL),A      ;2F19
77 AC5D CD0C00           CALL PRINTS
78 AC60 18C4             JR   VRFWTR      ;30CE
79 AC62          ;
80 AC62 16       TVERFY: DEFB 16H
81 AC63 56455249         DEFM 'VERIFY von: '
82 AC67 465920AB
83 AC6B B7B03A20
84 AC6F 0D               DEFB 0DH         ;3552
85 AC70 509D     TPRF:   DEFM 'Pr'
```

```
86 AC72 AD                  DEFB ADH
87 AC73 AAA4A5B3            DEFM 'fsumme: '
88 AC77 B3923A20
89 AC7B 0D                  DEFB 0DH              ;3B3E
90 AC7C             ;
91 AC7C                     END
```

22.3 OFFSET codiert Sprungweiten

Im Wörterbuch der Datentechnik finden wir unter «offset» die Übersetzung «Relativzeiger». Bei der indirekt-indizierten Adressierung (z. B. LD A,(IX+03)) kann man die 03 als Offset bezeichnen. Aber auch bei den relativen Sprüngen ist das, was wir bisher «Codierung der Sprungweite» nannten, ein Offset zum Inhalt des Programmzählers. Wir besprechen absichtlich erst jetzt ein Dienstprogramm für diese recht häufige Aufgabe.

P22.3.1 kommt ohne Arbeitsbereich aus und rechnet nur mit den Inhalten der Register: Die VON-Adresse wird nach HL geladen, die NACH-Adresse nach DE. HL wird zweimal inkrementiert und damit auf den aktuellen Programmzählerstand erhöht. Beide Register werden auf den Stapel gerettet, damit man ohne Datenverlust durch Subtraktion feststellen kann, ob ein Vorwärts- oder ein Rückwärtssprung zu berechnen ist. Vor der eigentlichen Verzweigung werden die Registerinhalte mit umgekehrter Zuordnung wiederhergestellt. Bei Rückwärtssprung wird gem. Erklärung von 5.2 die NACH-Adresse um 100h erhöht. Das geschieht durch Inkrementieren von H. Danach wird das Offset ermittelt. Ist danach H $\neq$ 0 oder Bit 7 von L gesetzt, würde der beabsichtigte Sprung die zulässige Weite überschreiten: Fehlermeldung.

```
P22.3.1   (Prüfsummen am rechten Rand)

13 AB47 11AAAB   OFFSET: LD   DE,TSPR         ;0166
14 AB4A CD1500           CALL MSG
15 AB4D CD0600   OFF2:   CALL LETNL
16 AB50 1129AE           LD   DE,TVON
17 AB53 CD1500           CALL MSG
18 AB56 CD57AF           CALL INBYTE          ;06B8
19 AB59 67               LD   H,A
```

```
20 AB5A CD57AF                CALL INBYTE
21 AB5D 6F                    LD   L,A
22 AB5E 11EEAC                LD   DE,TNACH
23 AB61 CD1500                CALL MSG              ;0BEE
24 AB64 CD57AF                CALL INBYTE
25 AB67 57                    LD   D,A
26 AB68 CD57AF                CALL INBYTE
27 AB6B 5F                    LD   E,A
28 AB6C 23                    INC  HL               ;106D
29 AB6D 23                    INC  HL
30 AB6E E5                    PUSH HL
31 AB6F D5                    PUSH DE
32 AB70 A7                    AND  A
33 AB71 ED52                  SBC  HL,DE            ;1430
34 AB73 E1                    POP  HL
35 AB74 D1                    POP  DE
36 AB75 FA88AB                JP   M,VORW
37 AB78 280E                  JR   Z,VORW
38 AB7A 24         RUECK:     INC  H                ;1869
39 AB7B A7                    AND  A
40 AB7C ED52                  SBC  HL,DE
41 AB7E 7C                    LD   A,H
42 AB7F A7                    AND  A
43 AB80 201D                  JR   NZ,LZUW          ;1BAF
44 AB82 CB7D                  BIT  7,L
45 AB84 2819                  JR   Z,LZUW
46 AB86 180B                  JR   LOFFS
47 AB88 A7         VORW:      AND  A
48 AB89 ED52                  SBC  HL,DE            ;1F41
49 AB8B 7C                    LD   A,H
50 AB8C A7                    AND  A
51 AB8D 2010                  JR   NZ,LZUW
52 AB8F CB7D                  BIT  7,L
53 AB91 200C                  JR   NZ,LZUW          ;2208
54 AB93 11C2AB     LOFFS:     LD   DE,TOFFS
55 AB96 CD1500                CALL MSG
56 AB99 7D                    LD   A,L
57 AB9A CD30AF                CALL OUTBYT
58 AB9D 18AE                  JR   OFF2             ;2757
59 AB9F CD0600     LZUW:      CALL LETNL
60 ABA2 11CCAB                LD   DE,TZUW
61 ABA5 CD1500                CALL MSG
62 ABA8 18A3                  JR   OFF2             ;2B4F
63 ABAA            ;
64 ABAA 16         TSPR:      DEFB 16H
65 ABAB 539E9DA5              DEFM 'Sprungweiten'
```

```
66 ABAF B097A392
67 ABB3 A69692B0
68 ABB7 9A929D92          DEFM 'berechnung'
69 ABBB 9F98B0A5
70 ABBF B097
71 ABC1 0D                DEFB 0DH             ;38CD
72 ABC2 204FAAAA TOFFS:   DEFM ' Offset: '
73 ABC6 A492963A
74 ABCA 20
75 ABCB 0D                DEFB 0DH             ;3CC3
76 ABCC A2A520A3 TZUW:    DEFM 'zu weit!!!'
77 ABD0 92A69621
78 ABD4 2121
79 ABD6 0D                DEFB 0DH             ;410B
80 ABD7          ;
81 ABD7                   END
```

22.4 MENUE schafft Überblick

Wir haben nun schon eine ganze Reihe von Utilities aufgestellt, und es erscheint sinnvoll, diese übersichtlich anzuordnen. Hierzu dient P22.4.1, das uns diese Übersicht auf den Bildschirm schreibt:

---

1. Disassembler
2. Offsetberechnung
3. Prüfsumme
4. Speicherauszug drucken
5. Umwandeln DEZ → HEX
6. Umwandeln HEX → DEZ
7. Verschieben
8. Verify
9. Monitor

Bild 22.1 Die Menü-Meldung auf dem Bildschirm

Danach können wir durch Drücken einer der Tasten 1 ... 9 das gesuchte Programm auswählen. Weil wir das Disassemblerprogramm erst in 23 aufstellen werden, sollten wir in B000 einstweilen C7 eintragen.

P22.4.1 (Prüfsummen am rechten Rand)

```
011 AA32 11A0AA     MENUE:  LD    DE,TMEN        ;015B
012 AA35 CDEDAF             CALL  MSGNL
013 AA38 11A8AA             LD    DE,T1
014 AA3B CDEDAF             CALL  MSGNL
015 AA3E 11B8AA             LD    DE,T2
016 AA41 CDEDAF             CALL  MSGNL          ;0B6C
017 AA44 11CCAA             LD    DE,T3
018 AA47 CDEDAF             CALL  MSGNL
019 AA4A 11D9AA             LD    DE,T4
020 AA4D CDEDAF             CALL  MSGNL
021 AA50 11F3AA             LD    DE,T5          ;1507
022 AA53 CDEDAF             CALL  MSGNL
023 AA56 110BAB             LD    DE,T6
024 AA59 CDEDAF             CALL  MSGNL
025 AA5C 1123AB             LD    DE,T7
026 AA5F CDEDAF             CALL  MSGNL          ;1DE8
027 AA62 1132AB             LD    DE,T8
028 AA65 CDEDAF             CALL  MSGNL
029 AA68 113CAB             LD    DE,T9
030 AA6B CDEDAF             CALL  MSGNL
031 AA6E CDB309     LOOP:   CALL  ??KEY          ;2629
032 AA71 FE21               CP    21H            ;'1'
033 AA73 CA00B0             JP    Z,DISASS
034 AA76 FE22               CP    22H            ;'2'
035 AA78 CA47AB             JP    Z,OFFSET
036 AA7B FE23               CP    23H            ;2CBF
037 AA7D CAB0AD             JP    Z,PRUESU
038 AA80 FE24               CP    24H            ;'4'
039 AA82 CAF5AC             JP    Z,PDUMP
040 AA85 FE25               CP    25H            ;'5'
041 AA87 CA34AE             JP    Z,DEZHEX       ;3542
042 AA8A FE26               CP    26H            ;'6'
043 AA8C CA92AE             JP    Z,HEXDEZ
044 AA8F FE27               CP    27H            ;'7'
045 AA91 CA7CAC             JP    Z,VERSCH
046 AA94 FE28               CP    28H            ;3CAD
047 AA96 CAD7AB             JP    Z,VERIFY
048 AA99 FE29               CP    29H            ;'9'
049 AA9B CA0000             JP    Z,MONITR
050 AA9E 18CE               JR    LOOP           ;41D0
051 AAA0            ;
052 AAA0 16         TMEN:   DEFB  16H
053 AAA1 4D92B0A5           DEFM  'Menue:'
054 AAA5 923A
055 AAA7 0D                 DEFB  0DH            ;44F3
```

```
056 AAA8 31292044 T1:      DEFM '1) Disassembler'
057 AAAC A6A4A1A4
058 AAB0 A492B39A
059 AAB4 B8929D
060 AAB7 0D                DEFB 0DH              ;4CB7
061 AAB8 3229204F T2:      DEFM '2) Offsetber'
062 AABC AAAAA492
063 AAC0 969A929D
064 AAC4 929F98B0          DEFM 'echnung'
065 AAC8 A5B097
066 AACB 0D                DEFB 0DH              ;56DC
067 AACC 33292050 T3:      DEFM '3) Pr'
068 AAD0 9D
069 AAD1 AD                DEFB ADH
070 AAD2 AAA4A5B3          DEFM 'fsumme'
071 AAD6 B392
072 AAD8 0D                DEFB 0DH              ;5CEA
073 AAD9 34292053 T4:      DEFM '4) Speichera'
074 AADD 9E92A69F
075 AAE1 98929DA1
076 AAE5 A5A4A2A5          DEFM 'uszug drucken'
077 AAE9 97209C9D
078 AAED A59FA992
079 AAF1 B0
080 AAF2 0D                DEFB 0DH              ;6A53
081 AAF3 35292055 T5:      DEFM '5) Umwandeln'
082 AAF7 B3A3A1B0
083 AAFB 9C92B8B0
084 AAFF 2044455A          DEFM ' DEZ -> HEX'
085 AB03 202D3E20
086 AB07 484558
087 AB0A 0D                DEFB 0DH              ;7303
088 AB0B 36292055 T6:      DEFM '6) Umwandeln'
089 AB0F B3A3A1B0
090 AB13 9C92B8B0
091 AB17 20484558          DEFM ' HEX -> DEZ'
092 AB1B 202D3E20
093 AB1F 44455A
094 AB22 0D                DEFB 0DH              ;7BB4
095 AB23 37292056 T7:      DEFM '7) Verschieben'
096 AB27 929DA49F
097 AB2B 98A6929A
098 AB2F 92B0
099 AB31 0D                DEFB 0DH              ;82B5
100 AB32 38292056 T8:      DEFM '8) Verify'
101 AB36 929DA6AA
```

```
102 AB3A BD
103 AB3B 0D                  DEFB 0DH              ;86D5
104 AB3C 3929204D T9:        DEFM '9) Monitor'
105 AB40 B7B0A696
106 AB44 B79D
107 AB46 0D                  DEFB 0DH              ;8BA8
108 AB47            ;
109 AB47                     END
```

23
Disassembler

Ein Disassembler ist ein Hilfsprogramm, das ein im Speicher stehendes Maschinenprogramm (Objektcode) teilweise in die Assemblersprache (Mnemonics, Operanden usw.) zurückübersetzt. Das wird häufig bei der Fehlersuche so gemacht und bei der Ausdeutung fremder Programme, deren Quelltext nicht vorliegt. Da beim Assemblieren die Programmiervariablen (Labels, symbolische Adressen usw.) nicht mit ihrem Namen in das Maschinenprogramm übergeben werden, sind diese nicht im Objektcode vorhanden und können auch aus ihm nicht zurückgewonnen werden. Beim Disassemblieren erhält man also nur die Werte der Programmiervariablen und nicht ihre Namen. Man bezeichnet diese Darstellungsweise auch als Disassemblerformat. Ein Beispiel soll das deutlich machen: Im Quelltext können wahlweise Dezimal- oder Hexadezimalzahlen (in manchen Assemblern auch Binärzahlen) stehen. Der Disassembler erfährt davon nichts und liefert grundsätzlich Hexadezimalzahlen:

```
Assembler-Quelltext (so wird eingegeben)

PRNT:EQU 0012H
REL 200H
LABEL:LD A,51
CALL PRNT
JR LABEL
END

Assemblerprotokoll (im Assemblerformat)

0000 P          PRNT:    EQU   0012H
0000                     REL   2000H
2000 3E33       LABEL:   LD    A,51
2002 CD1200              CALL  PRNT
2005 18F9                JR    LABEL
2007                     END
```

```
Maschinenprogramm (Objektcode)

2000 3E 33 CD 12 00 18 F9


Dasselbe Programm im Disassemblerformat

2000 3E 33          LD   A,33
2002 CD 12 00       CALL 1200
2005 18 F9          JR   2000
```

23.1 Zielsetzung und Grobentwurf

Wir wollen kurz vor Abschluß dieses Buches einen Disassembler schreiben. Zielsetzung:

1. Das Programm muß voll funktionsfähig sein.
2. Es soll lesbar sein.
3. Die Bedienung soll so aussehen:

Das Programm meldet sich als Disassembler. Auf die Frage «von» wird die Anfangsadresse des zu disassemblierenden Speicherbereichs eingegeben, auf die Frage «bis» die Endadresse und auf die Frage «D/P» ein D, wenn nur angezeigt werden soll, oder ein P, wenn die Ausgabe auf Printer und Display erfolgen soll.

Das Ausgabeformat wurde schon in Abschnitt 21.2 umrissen; die Ausgabezeilen sollen in 1200...1227 aufgebaut werden und müssen mit 0D enden, denn sie sollen mit MSG (bzw. und PMSG) ausgegeben werden.

Wir teilen das Programm teils als verkürztes Assemblerprotokoll mit, teils als Hexdump. Weitere Erkenntnisse kann der Leser gewinnen, indem er den Disassembler sich selbst disassemblieren läßt. Noch eine Verständnishilfe: Das IX-Register enthält stets die Anfangsadresse des jeweils zu disassemblierenden Befehls.

```
P23.1.1 Speicherdisposition / Arbeitsbereich

0000 P          ABER:    EQU  1200H     ;1200...1227
0000 P          VON:     EQU  ABER+28H  ;1228...1229
0000 P          BIS:     EQU  VON+02    ;122A...122B
0000 P          D/P:     EQU  BIS+02    ;122C
0000 P          EBER:    EQU  D/P+01    ;122D...122E
0000 P          JUMP:    EQU  EBER+02   ;122F...1230
0000 P          MNEMAD:  EQU  JUMP+02   ;1231...1232
0000 P          OPCODE:  EQU  MNEMAD+02;1233
0000 P          SYMBOL:  EQU  OPCODE+01;1234
0000 P          XYFLAG:  EQU  SYMBOL+01;1235
```

23.2 Die trivialen Anfangsroutinen

Das Hauptprogramm und die trivialen Anfangsroutinen können später durch Disassemblieren verständlich gemacht werden:

```
P23.2.1
DISASS B000 11 08 B7 CD 15 00 CD 06 00 CD
       B00A 20 B0 CD 5A B0 C3 9C B0            0808
WEITER B012 A7 ED 5B 28 12 2A 2A 12 ED 52
       B01C 38 EB 18 EC                        0DFD
NEUBLK B020 11 25 B7 CD 15 00 21 29 12 CD
       B02A 4C AF 11 2A B7 CD 15 00 21 2B      1410
       B034 12 CD 4C AF 3E 00 32 2C 12 11
       B03E 30 B7 CD 15 00 CD B3 09 CD CE      1B96
       B048 0B F5 CD 12 00 CD 06 00 F1 FE
       B052 50 C0 3E 01 32 2C 12 C9            22BF
NEUZEI B05A CD 8A B0 CD 65 B0 DD 2A 28 12
       B064 C9                                 28B2
ZEINR  B065 21 00 12 ED 5B 28 12 CD 74 B0
       B06F 53 CD 74 B0 C9                     2F65
```

Die Routine D/ASC verwandelt eine im D-Register stehende zweistellige Hexadezimalzahl (also ein Byte) in den ASCII und legt die beiden Bytes in (HL) ab.

```
P23.2.2
D/ASC  B074 7A F5 CB 3F CB 3F CB 3F CB 3F
       B07E CD DA 03 77 23 F1 CD DA 03 77
       B088 23 C9                              0BD9
```

```
NZINIT B08A 21 00 12 06 17 36 20 23 10 FB
       B094 06 11 36 0D 23 10 FB C9              0FFE
```

Dazu gehören die Texte

```
TTITL  B708 16 44 A6 A4 A1 A4 A4 92 B3 9A
       B712 B8 92 9D 0D                          0760
TFEHL  B716 2A 2A 20 46 92 98 B8 92 9D 3F
       B720 20 2A 2A 20 0D                       0C0B
TVON   B725 AB B7 B0 20 0D
TBIS   B72A 20 9A A6 A4 20 0D
TD/P   B730 20 44 2F 50 20 0D                    118B
```

23.3 Einteilung und Codierung der Befehle

Bevor wir zur ersten wesentlichen Decodierungsroutine gelangen, müssen wir auf die Klassifizierung der 698 Befehle der Z80-CPU eingehen. Hierfür sind nicht funktionale, sondern lediglich drucktechnische Gesichtspunkte und die benötigten Decodierungsroutinen maßgebend:

Gruppe 1

z. B. 00	NOP
oder 02	LD (BC), A

33 1-Byte-Befehle, denen unmittelbar Mnemonics zugeordnet werden:

```
P23.3.1
B736 35B2      TGR1:    DEFW SGR1
B738 21                 DEFB 21H                ;Anzahl
B739 00                 DEFB 00
B73A 02                 DEFB 02
.... ..                 .... ....
B75A 4E4F50    UGR1:    DEFM 'NOP'
B75D 0D                 DEFB 0DH
B75E 4C442020           DEFM 'LD    (BC),A'
B762 20284243
B766 292C41
B769 0D                 DEFB 0DH
.... ........           .... ....
```

Dasselbe vollständig als Hexdump:

```
TGR1   B736 35 B2 21 00 02 07 08 0A 0F 12
       B740 17 1A 1F 27 2F 34 35 37 3F 76        033F
```

```
         B74A 86 8E 96 9E A6 AE B6 BE C9 D9
         B754 E3 E9 EB F3 F9 FB                  0F8F
UGR1     B75A 4E 4F 50 0D 4C 44 20 20 20 28
         B764 42 43 29 2C 41 0D 52 4C 43 41      13EB
         B76E 0D 45 58 20 20 20 41 46 2C 41
         B778 46 27 0D 4C 44 20 20 41 2C 28      17C8
         B782 42 43 29 0D 52 52 43 41 0D 4C
         B78C 44 20 20 20 28 44 45 29 2C 41      1BEF
         B796 0D 52 4C 41 0D 4C 44 20 20 20
         B7A0 41 2C 28 44 45 29 0D 52 52 41      2011
         B7AA 0D 44 41 41 0D 43 50 4C 0D 49
         B7B4 4E 43 20 20 28 48 4C 29 0D 44      242D
         B7BE 45 43 20 20 28 48 4C 29 0D 53
         B7C8 43 46 0D 43 43 46 0D 48 41 4C      287E
         B7D2 54 0D 41 44 44 20 20 41 2C 28
         B7DC 48 4C 29 0D 41 44 43 20 20 41      2C90
         B7E6 2C 28 48 4C 29 0D 53 55 42 20
         B7F0 20 28 48 4C 29 0D 53 42 43 20      30C2
         B7FA 20 41 2C 28 48 4C 29 0D 41 4E
         B804 44 20 20 28 48 4C 29 0D 58 4F      34ED
         B80E 52 20 20 28 48 4C 29 0D 4F 52
         B818 20 20 20 28 48 4C 29 0D 43 50      38F7
         B822 20 20 20 28 48 4C 29 0D 52 45
         B82C 54 0D 45 58 58 0D 45 58 20 20      3D20
         B836 20 28 53 50 29 2C 48 4C 0D 4A
         B840 50 20 20 20 28 48 4C 29 0D 45      4132
         B84A 58 20 20 20 44 45 2C 48 4C 0D
         B854 44 49 0D 4C 44 20 20 20 53 50
         B85E 2C 48 4C 0D 45 49 0D               46D5
```

Gruppe 2

z. B. 00 . . 0011 INC ss

3 ⋆ 4 = 12 1-Byte-Befehle, die durch Maskieren mit CF identifiziert werden, und in denen ein 16-bit-Register der Gruppe ss (BC, DE, HL, SP) zu decodieren ist. Gruppe 2A entsprechend mit Registergruppe qq (BC, DE, HL, AF):

```
P23.3.2
TGR2     B865 3B B2 03 03 09 0B
UGR2     B86B 49 4E 43 20 20 A4 A4 0D 41 44      03FB
         B875 44 20 20 48 4C 2C A4 A4 0D 44
         B87F 45 43 20 20 A4 A4 0D               08F5
TGR2A    B886 44 B2 02 C1 C5
UGR2A    B88B 50 4F 50 20 20 A0 A0 0D 50 55
         B895 53 48 20 A0 A0 0D                  109C
```

Gruppe 3
z. B. 00 . . . 100 INC r

Insgesamt 12 * 7 = 84 1-Byte-Befehle, die durch Maskieren mit C7 identifiziert werden, und in denen ein 8-bit-Register (B, C, D, E, H, L, A) zu decodieren ist. Dem Code 110 entspräche (HL), was jedoch vier Schreibpositionen belegt und deshalb schon in Gruppe 1 vorwegzunehmen war.

```
P23.3.3
TGR3    B89B 4D B2 03 04 05 46
UGR3    B8A1 49 4E 43 20 20 9D 0D 44 45 43     03E1
        B8AB 20 20 9D 0D 4C 44 20 20 20 9D
        B8B5 2C 28 48 4C 29 0D                 0776
TGR3A   B8BB 56 B2 09 70 80 88 90 98 A0 A8
        B8C5 B0 B8                             0DD7
UGR3A   B8C7 4C 44 20 20 20 28 48 4C 29 2C
        B8D1 9D 0D 41 44 44 20 20 41 2C 9D     1295
        B8DB 0D 41 44 43 20 20 41 2C 9D 0D
        B8E5 53 55 42 20 20 9D 0D 53 42 43     176D
        B8EF 20 20 41 2C 9D 0D 41 4E 44 20
        B8F9 20 9D 0D 58 4F 52 20 20 9D 0D     1C64
        B903 4F 52 20 20 20 9D 0D 43 50 20
        B90D 20 20 9D 0D                       1FAC
```

Gruppe 4
01 . . . . . . LD r,r'

49 1-Byte-Befehle, die durch Maskieren mit C0 identifiziert werden, und in denen zwei Register der Gruppe r zu dekodieren sind. (HL) bleibt wieder ausgespart. Insbesondere wäre formal 76 = 0111 0110 = 01 110 110 = LD (HL),(HL) eine unzulässige Deutung. Die Gruppe 1 mit dem Befehl HALT muß deshalb schon vorher abgearbeitet werden.

```
P23.3.4
TGR4    B911 5F B2 01 40
UGR4    B915 4C 44 20 20 20 9D 2C 9D 0D        03B5
```

Gruppe 5
11 . . . 000 RET cc

Gruppe 5A
11 . . . 111 RST tt

```
P23.3.5
TGR5    B91E 6B B2 01 C0
UGR5    B922 52 45 54 20 20 9F 9F 0D               0454
TGR5A   B92A 74 B2 01 C7
UGR5A   B92E 52 53 54 20 20 96 96 0D               08B4
```

Gruppe 6
z. B. 36 nn LD (HL),nn

11 2-Bytes-Befehle; das erste Byte bestimmt den Mnemonic, das zweite wird einfach angehängt. Abweichend von der üblichen Befehlsnotierung bedeutet hier jedes n eine Schreibposition, also statt eines Bytes nur ein Halbbyte.

```
P23.3.6
TGR6    B936 7D B2 0B 36 C6 CE D3 D6 DB DE
        B940 E6 EE F6 FE                           0A2E
UGR6    B944 4C 44 20 20 20 28 48 4C 29 2C
        B94E B0 B0 0D 41 44 44 20 20 41 2C         0F12
        B958 B0 B0 0D 41 44 43 20 20 41 2C
        B962 B0 B0 0D 4F 55 54 20 20 28 B0         1571
        B96C B0 29 2C 41 0D 53 55 42 20 20
        B976 B0 B0 0D 49 4E 20 20 20 41 2C         1ABF
        B980 28 B0 B0 29 0D 53 42 43 20 20
        B98A 41 2C B0 B0 0D 41 4E 44 20 20         2082
        B994 B0 B0 0D 58 4F 52 20 20 B0 B0
        B99E 0D 4F 52 20 20 20 B0 B0 0D 43
        B9A8 50 20 20 B0 B0 0D                     2943
```

Gruppe 7
z. B. 10 ee DJNZ aaaa

6 2-Bytes-Befehle, bei denen eine Sprungweite in eine absolute Adresse umzurechnen ist.

```
P23.3.7
TGR7    B9AE 96 B2 06 10 18 20 28 30 38
UGR7    B9B7 44 4A 4E 5A 20 B0 B0 B0 B0 0D         0649
        B9C1 4A 52 20 20 20 B0 B0 B0 B0 0D
        B9CB 4A 52 20 20 20 4E 5A 2C B0 B0         0D42
        B9D5 B0 B0 0D 4A 52 20 20 20 5A 20
        B9DF 2C B0 B0 B0 B0 0D 4A 52 20 20         13FA
        B9E9 20 4E 43 2C B0 B0 B0 B0 0D 4A
        B9F3 52 20 20 20 43 20 2C B0 B0 B0
        B9FD B0 0D                                 1BCF
```

Gruppe 8
00 . . . 110 nn LD r,nn

```
P23.3.8
TGR8    B9FF C5 B2 01 06
UGR8    BA03 4C 44 20 20 20 9D 2C B0 B0 0D        04A4
```

Gruppe 9
z. B. 22 nn nn LD (nnnn),HL

```
P23.3.9
TGR9    BA0D E2 B2 06 22 2A 32 3A C3 CD
UGR9    BA16 4C 44 20 20 20 28 B0 B0 B0 B0     07BA
        BA20 29 2C 48 4C 0D 4C 44 20 20 20
        BA2A 48 4C 2C 28 B0 B0 B0 B0 29 0D     0D7E
        BA34 4C 44 20 20 20 28 B0 B0 B0 B0
        BA3E 29 2C 41 0D 4C 44 20 20 20 41     132A
        BA48 2C 28 B0 B0 B0 B0 29 0D 4A 50
        BA52 20 20 20 B0 B0 B0 B0 0D 43 41
        BA5C 4C 4C 20 B0 B0 B0 B0 0D           1E44
```

Gruppe 10
00 . . 0001 nn nn LD ss,nnnn

```
P23.3.10
TGR10   BA64 0A B3 01 01
UGR10   BA68 4C 44 20 20 20 A4 A4 2C B0 B0
        BA72 B0 B0 0D                          05F0
```

Gruppe 11
11 . . . 010 nn nn JP cc,nnnn

```
P23.3.11
TGR11   BA75 13 B3 02 C2 C4
UGR11   BA7A 4A 50 20 20 20 9F 9F 2C B0 B0     0612
        BA84 B0 B0 0D 43 41 4C 4C 20 9F 9F
        BA8E 2C B0 B0 B0 B0 0D                 0CF2
```

Gruppe C1
z. B. CB 06 RLC (HL)

```
P23.3.12
TGRC1   BA94 1C B3 07 06 0E 16 1E 26 2E 3E
UGRC1   BA9E 52 4C 43 20 20 28 48 4C 29 0D        03C3
        BAA8 52 52 43 20 20 28 48 4C 29 0D
        BAB2 52 4C 20 20 20 28 48 4C 29 0D        07CC
        BABC 52 52 20 20 20 28 48 4C 29 0D
        BAC6 53 4C 41 20 20 28 48 4C 29 0D        0BD4
        BAD0 53 52 41 20 20 28 48 4C 29 0D
        BADA 53 52 4C 20 20 28 48 4C 29 0D        100F
```

Gruppe C2

z. B. CB 00000 . . .   RLC r

```
P23.3.13
TGRC2   BAE4 22 B3 07 00 08 10 18 20 28 38
UGRC2   BAEE 52 4C 43 20 20 9D 0D 52 52 43        043E
        BAF8 20 20 9D 0D 52 4C 20 20 20 9D
        BB02 0D 52 52 20 20 20 9D 0D 53 4C        091D
        BB0C 41 20 20 9D 0D 53 52 41 20 20
        BB16 9D 0D 53 52 4C 20 20 9D 0D           0DF3
```

Gruppe C3

z. B. CB 01 . . . 110   BIT b,(HL)

Liest man die Belegung der drei Punkte binär, erhält man den Index des adressierten Bits.

```
P23.3.14
TGRC3   BB1F 31 B3 03 46 86 C6
UGRC3   BB25 42 49 54 20 20 9A 2C 28 48 4C        051A
        BB2F 29 0D 52 45 53 20 20 9A 2C 28
        BB39 48 4C 29 0D 53 45 54 20 20 9A
        BB43 2C 28 48 4C 29 0D                    0B16
```

Gruppe C4

z. B. CB 01 . . . . . .   BIT b,r

```
P23.3.15
TGRC4   BB49 3A B3 03 40 80 C0
UGRC4   BB4F 42 49 54 20 20 9A 2C 9D 0D 52        0551
        BB59 45 53 20 20 9A 2C 9D 0D 53 45
        BB63 54 20 20 9A 2C 9D 0D                 0A35
```

Gruppe E1
z. B. ED 44 NEG

```
P23.3.16
TGRE1   BB6A 46 B3 1C 44 45 46 47 4D 4F 56
        BB74 57 5E 5F 67 6F A0 A1 A2 A3 A8        0835
        BB7E A9 AA AB B0 B1 B2 B3 B8 B9 BA
        BB88 BB                                   0FDF
UGRE1   BB89 4E 45 47 0D 52 45 54 4E 0D 49
        BB93 4D 20 20 20 30 0D 4C 44 20 20        140F
        BB9D 20 49 2C 41 0D 52 45 54 49 0D
        BBA7 4C 44 20 20 20 52 2C 41 0D 49        1838
        BBB1 4D 20 20 20 31 0D 4C 44 20 20
        BBBB 20 41 2C 49 0D 49 4D 20 20 20        1BCC
        BBC5 32 0D 4C 44 20 20 20 41 2C 52
        BBCF 0D 52 52 44 0D 52 4C 44 0D 4C        1FF7
        BBD9 44 49 0D 43 50 49 0D 49 4E 49
        BBE3 0D 4F 55 54 49 0D 4C 44 44 0D        2496
        BBED 43 50 44 0D 49 4E 44 0D 4F 55
        BBF7 54 44 0D 4C 44 49 52 0D 43 50        2976
        BC01 49 52 0D 49 4E 49 52 0D 4F 54
        BC0B 49 52 0D 4C 44 44 52 0D 43 50        2E6E
        BC15 44 52 0D 49 4E 44 52 0D 4F 54
        BC1F 44 52 0D                             3191
```

Gruppe E2
z. B. ED 01 . . . 000 IN r,(C)

```
P23.3.17
TGRE2   BC22 4C B3 02 40 41
UGRE2   BC27 49 4E 20 20 20 9D 2C 28 43 29        03D6
        BC31 0D 4F 55 54 20 20 28 43 29 2C
        BC3B 9D 0D                                0685
```

Gruppe E3
z. B. ED 01 . . 0010 SBC HL,ss

Gruppe E4
z. B. ED 01 . .0011 nn nn LD (nnnn),ss

```
P23.3.18
TGRE3   BC3D 5B B3 02 42 4A
UGRE3   BC42 53 42 43 20 20 48 4C 2C A4 A4        04BC
        BC4C 0D 41 44 43 20 20 48 4C 2C A4
        BC56 A4 0D                                07E6
```

```
TGRE4   BC58 6A B3 02 43 4B
UGRE4   BC5D 4C 44 20 20 20 28 B0 B0 B0 B0
        BC67 29 2C A4 A4 0D 4C 44 20 20 20      0D6B
        BC71 A4 A4 2C 28 B0 B0 B0 B0 29 0D      1497
```

Gruppe D1

z. B. DD 23 INC IX

bzw. FD 23 INC IY

Diese Gruppe (und die folgenden) schließt auch die Befehle ein, bei denen nach der Decodierung lediglich ein (oder zwei) X in Y umzuwandeln ist.

```
P23.3.19
TGRD1   BC7B 84 B3 07 23 2B E1 E3 E5 E9 F9
UGRD1   BC85 49 4E 43 20 20 49 58 0D 44 45      0868
        BC8F 43 20 20 49 58 0D 50 4F 50 20
        BC99 20 49 58 0D 45 58 20 20 20 28      0C9B
        BCA3 53 50 29 2C 49 58 0D 50 55 53
        BCAD 48 20 49 58 0D 4A 50 20 20 20      1149
        BCB7 28 49 58 29 0D 4C 44 20 20 20
        BCC1 53 50 2C 49 58 0D                  14B5
TGRD2   BCC7 8D B3 01 09
UGRD2   BCCB 41 44 44 20 20 49 58 2C 9E 9E
        BCD5 0D                                 191E
```

Die restlichen Gruppen werden ohne nähere Spezifikation mitgeteilt:

```
P23.3.20
TGRD3   BCD6 99 B3 0A 34 35 86 8E 96 9E A6
        BCE0 AE B6 BE                           06CF
UGRD3   BCE3 49 4E 43 20 20 28 49 58 2B 9C
        BCED 9C 29 0D 44 45 43 20 20 28 49      0BC8
        BCF7 58 2B 9C 9C 29 0D 41 44 44 20
        BD01 20 41 2C 28 49 58 2B 9C 9C 29      1184
        BD0B 0D 41 44 43 20 20 41 2C 28 49
        BD15 58 2B 9C 9C 29 0D 53 55 42 20      1672
        BD1F 20 28 49 58 2B 9C 9C 29 0D 53
        BD29 42 43 20 20 41 2C 28 49 58 2B      1B6D
        BD33 9C 9C 29 0D 41 4E 44 20 20 28
        BD3D 49 58 2B 9C 9C 29 0D 58 4F 52      2149
        BD47 20 20 28 49 58 2B 9C 9C 29 0D
        BD51 4F 52 20 20 20 28 49 58 2B 9C      267C
        BD5B 9C 29 0D 43 50 20 20 20 28 49
        BD65 58 2B 9C 9C 29 0D                  2AA3
```

```
TGRD4   BD6B A5 B3 01 46
UGRD4   BD6F 4C 44 20 20 20 9D 2C 28 49 58
        BD79 2B 9C 9C 29 0D                       305D
TGRD4A  BD7E B4 B3 01 70
UGRD4A  BD82 4C 44 20 20 20 28 49 58 2B 9C
        BD8C 9C 29 2C 9D 0D                       3650
TGRD5   BD91 C3 B3 01 36
UGRD5   BD95 4C 44 20 20 20 28 49 58 2B 9C
        BD9F 9C 29 2C B0 B0 0D                    3CDB
TGRD6   BDA5 DA B3 03 21 22 2A
UGRD6   BDAB 4C 44 20 20 20 49 58 2C B0 B0        41F5
        BDB5 B0 B0 0D 4C 44 20 20 20 28 B0
        BDBF B0 B0 B0 29 2C 49 58 0D 4C 44        48CD
        BDC9 20 20 20 49 58 2C 28 B0 B0 B0
        BDD3 B0 29 0D                             4D18
TGRDC1  BDD6 F4 B3 07 06 0E 16 1E 26 2E 3E
UGRDC1  BDE0 52 4C 43 20 20 28 49 58 2B 9C        5251
        BDEA 9C 29 0D 52 52 43 20 20 28 49
        BDF4 58 2B 9C 9C 29 0D 52 4C 20 20        578A
        BDFE 20 28 49 58 2B 9C 9C 29 0D 52
        BE08 52 20 20 20 28 49 58 2B 9C 9C        5D3C
        BE12 29 0D 53 4C 41 20 20 28 49 58
        BE1C 2B 9C 9C 29 0D 53 52 41 20 20        621A
        BE26 28 49 58 2B 9C 9C 29 0D 53 52
        BE30 4C 20 20 28 49 58 2B 9C 9C 29
        BE3A 0D                                   680F
TGRDC2  BE3B 00 B4 03 46 86 C6
UGRDC2  BE41 42 49 54 20 20 9A 2C 28 49 58        6D06
        BE4B 2B 9C 9C 29 0D 52 45 53 20 20
        BE55 9A 2C 28 49 58 2B 9C 9C 29 0D        72F1
        BE5F 53 45 54 20 20 9A 2C 28 49 58
        BE69 2B 9C 9C 29 0D                       7745
```

Dann fehlen nur noch die einzusetzenden Symbole:

```
P23.3.21
Tb      BE6E 30 31 32 33 34 35 36 37
Tcc     BE76 4E 5A 5A 20 4E 43 43 20 50 4F
        BE80 50 45 50 20 4D 20                    05C3
Tpp     BE86 42 43 44 45 49 58 53 50
Tqq     BE8E 42 43 44 45 48 4C 41 46              0A3E
Tr      BE96 42 43 44 45 48 4C 2A 41
Tss     BE9E 42 43 44 45 48 4C 53 50              0E90
Ttt     BEA6 30 30 30 38 31 30 31 38 32 30
        BEB0 32 38 33 30 33 38                    11BC

Prüfsumme (B708 ... BEB5) = 23233
```

23.4 Die ersten Decodierungsroutinen

Wenn man trotz der Vielfalt der Z80-Befehle zeitsparend disassemblieren will, muß man frühzeitig und effektiv in Befehlsgruppen verzweigen. Diese Aufgabe erfüllt DECODE nebst nachfolgenden Programmabschnitten. Danach werden IY und HL mit den Anfängen der Tabellen TGR und UGR geladen und eine SUchMNemonic-Routine aufgerufen.

```
P23.4.1
B09C DD7E00    DECODE: LD    A,(IX+00)
B09F 323312            LD    (OPCODE),A
B0A2 FECB              CP    CBH
B0A4 CA45B1            JP    Z,GRCB
B0A7 FEED              CP    EDH
B0A9 CA76B1            JP    Z,GRED
B0AC FEDD              CP    DDH
B0AE CAA7B1            JP    Z,GRDD
B0B1 FEFD              CP    FDH
B0B3 CAAEB1            JP    Z,GRFD              ;1158
B0B6 FD2136B7          LD    IY,TGR1             ;Gr.1
B0BA 215AB7            LD    HL,UGR1
B0BD CD0FB4            CALL  SUMNEM
B0C0 FD2165B8          LD    IY,TGR2             ;Gr.2
B0C4 216BB8            LD    HL,UGR2
B0C7 CD3EB4            CALL  SUMN02
B0CA FD2186B8          LD    IY,TGR2A            ;Gr.2A
B0CE 218BB8            LD    HL,UGR2A
B0D1 CD53B4            CALL  SUMN2A              ;20F7
Und weiter als Hexdump
          B0D4 FD 21 9B B8 21 A1 B8 CD 63 B4
          B0DE FD 21 BB B8 21 C7 B8 CD 78 B4      0BF9
          B0E8 FD 21 11 B9 21 15 B9 CD 8F B4
          B0F2 FD 21 1E B9 21 22 B9 CD 9A B4      15EC
          B0FC FD 21 2A B9 21 2E B9 CD A5 B4
          B106 FD 21 36 B9 21 44 B9 CD 0F B4      1FD6
          B110 FD 21 AE B9 21 B7 B9 CD 0F B4
          B11A FD 21 FF B9 21 03 BA CD B0 B4      2B61
          B124 FD 21 0D BA 21 16 BA CD 0F B4
          B12E FD 21 64 BA 21 68 BA CD BB B4      3582
          B138 FD 21 75 BA 21 7A BA CD C6 B4
          B142 C3 EF B6                           3DD3
GRCB      B145 DD 7E 01 32 33 12 FD 21 94 BA
          B14F 21 9E BA CD 12 B4 FD 21 E4 BA      47DA
          B159 21 EE BA CD D6 B4 FD 21 1F BB
```

```
        B163 21 25 BB CD EB B4 FD 21 49 BB      5381
        B16D 21 4F BB CD 00 B5 C3 EF B6
GRED    B176 DD 7E 01 32 33 12 FD 21 6A BB      5CAC
        B180 21 89 BB CD 12 B4 FD 21 22 BC
        B18A 21 27 BC CD 15 B5 FD 21 3D BC      6652
        B194 21 42 BC CD 25 B5 FD 21 58 BC
        B19E 21 5D BC CD 35 B5 C3 EF B6         70A3
GRDD    B1A7 3E 00 32 35 12 18 05
GRFD    B1AE 3E 01 32 35 12
GRDDFD  B1B3 DD 7E 01 32 33 12 FE CB CA 07      769C
        B1BD B2 FD 21 7B BC 21 85 BC CD 0F
        B1C7 B4 FD 21 C7 BC 21 CB BC CD 45      81F0
        B1D1 B5 FD 21 D6 BC 21 E3 BC CD 0F
        B1DB B4 FD 21 6B BD 21 6F BD CD 50      8D55
        B1E5 B5 FD 21 7E BD 21 82 BD CD 5B
        B1EF B5 FD 21 91 BD 21 95 BD CD 0F      985B
        B1F9 B4 FD 21 A5 BD 21 AB BD CD 0F
        B203 B4 C3 EF B6                        A110
GRDCFC  B207 DD 7E 03 32 33 12 E6 C7 FE 46
        B211 28 15 FE 86 28 11 FE C6 28 0D      A9C9
        B21B FD 21 D6 BD 21 E0 BD CD 0F B4
        B225 C3 EF B6 FD 21 3B BE 21 41 BE
        B22F CD 12 B4 C3 EF B6                  B962
```

SGR sind Subroutinen, die die Bearbeitung der einzelnen Gruppen von einer ziemlich hohen Ebene aus steuern. Zuerst wird der gefundene Mnemonic an die richtige Stelle des Ausgabebereichs übertragen (BLockTRAnSfer). Dann wird IX um eins erhöht und gleichzeitig ein Byte (Opcode) in den Ausgabebereich übertragen. Im Verlauf von SGR2 wird außerdem noch in den Schriftsatz für ss das durch Bit5 bis Bit4 bestimmte Symbol hineingetauscht (EXchange). So erledigt jede SGR-Routine ihre Koordinierungsaufgabe.

```
P23.4.2
B235 CD66B5    SGR1:    CALL BLTRAS
B238 C384B5             JP   INKR1
B23B CD66B5    SGR2:    CALL BLTRAS
B23E CD8BB6             CALL EXss54
B241 C384B5             JP   INKR1
B244 CD66B5    SGR2A:   CALL BLTRAS
B247 CD64B6             CALL EXqq54
B24A C384B5             JP   INKR1             ;0FA1
```

Und weiter als Hexdump

```
SGR3    B24D CD 66 B5 CD 71 B6 C3 84 B5
SGR3A   B256 CD 66 B5 CD 7E B6 C3 84 B5           1B5E
SGR4    B25F CD 66 B5 CD 71 B6 CD 7E B6 C3
        B269 84 B5                                2337
SGR5    B26B CD 66 B5 CD 4A B6 C3 84 B5
SGR5A   B274 CD 66 B5 CD 98 B6 C3 84 B5           2EE7
SGR6    B27D CD 66 B5 DD 56 01 D5 21 08 12
        B287 CD 74 B0 D1 3E B0 32 34 12 CD        3808
        B291 36 B6 C3 7F B5
SGR7    B296 CD 66 B5 DD 56 01 D5 21 08 12        3F17
        B2A0 CD 74 B0 D1 5A 16 00 DD E5 E1
        B2AA CB 7B 28 01 25 23 23 19 54 5D        4790
        B2B4 D5 3E B0 32 34 12 CD 36 B6 D1
        B2BE 53 CD 36 B6 C3 7F B5                 5058
SGR8    B2C5 CD 66 B5 CD 71 B6 DD 56 01 21
        B2CF 08 12 CD 74 B0 3E B0 32 34 12        58FA
        B2D9 DD 56 01 CD 36 B6 C3 7F B5
SGR9    B2E2 CD 66 B5 DD 56 01 21 08 12 CD        6202
        B2EC 74 B0 DD 56 02 D5 21 0B 12 CD
        B2F6 74 B0 D1 3E B0 32 34 12 CD 36        6A99
        B300 B6 DD 56 01 CD 36 B6 C3 7A B5
SGR10   B30A CD 66 B5 CD 8B B6 C3 E5 B2
SGR11   B313 CD 66 B5 CD 4A B6 C3 E5 B2           7C8D
SGRC1   B31C CD 66 B5 C3 7F B5
SGRC2   B322 CD 66 B5 DD 7E 01 32 33 12 CD        84F4
        B32C 7E B6 C3 7F B5 CD 66 B5 CD 3D
        B336 B6 C3 7F B5                          8DBE
SGRC4   B33A CD 66 B5 CD 3D B6 CD 7E B6 C3
        B344 7F B5                                955E
SGRE1   B346 CD 66 B5 C3 7F B5
SGRE2   B34C CD 66 B5 DD 7E 01 32 33 12 CD        9DC5
        B356 71 B6 C3 7F B5
SGRE3   B35B CD 66 B5 DD 7E 01 32 33 12 CD        A56B
        B365 8B B6 C3 7F B5
SGRE4   B36A CD 66 B5 CD 8B B6 3E B0 32 34        ADED
        B374 12 DD 56 03 CD 36 B6 DD 56 02
        B37E CD 36 B6 C3 75 B5                    B5C9
SGRD1   B384 CD 66 B5 CD B2 B6 C3 7F B5
SGRD2   B38D CD 66 B5 CD 57 B6 CD B2 B6 C3        C297
        B397 7F B5
SGRD3   B399 CD 66 B5 CD B2 B6 CD C9 B6 C3        CAF7
        B3A3 7A B5
SGRD4   B3A5 CD 66 B5 CD B2 B6 CD C9 B6 CD        D35C
        B3AF 71 B6 C3 7A B5
```

```
SGRD4A  B3B4 CD 66 B5 CD B2 B6 CD C9 B6 CD        DDAB
        B3BE 7E B6 C3 7A B5
SGRD5   B3C3 CD 66 B5 CD B2 B6 CD C9 B6 3E        E778
        B3CD B0 32 34 12 DD 56 03 CD 36 B6
        B3D7 C3 75 B5                             ED7C
SGRD6   B3DA CD 66 B5 CD B2 B6 3E B0 32 34
        B3E4 12 DD 56 03 CD 36 B6 DD 56 02        F723
        B3EE CD 36 B6 C3 75 B5
SGRDC1  B3F4 CD 66 B5 CD B2 B6 CD C9 B6 C3        101F5
        B3FE 75 B5
SGRDC2  B400 CD 66 B5 CD B2 B6 CD C9 B6 CD
        B40A 3D B6 C3 75 B5                       10D35
```

23.5 Das Auffinden der Mnemonics

Die Subroutine SUMNEM (SUche MNEMonic) und die folgenden übertragen zunächst die in TGR stehende SGR-Adresse nach JUMP und laden die Anzahl der Eintragungen in TGR/UGR nach B. Beim Suchen wird in TGR immer um eins weitergezählt, in UGR bis zum nächsten 0D. Die Adresse eines als richtig erkannten Mnemonics wird nach MNEMAD zwischengespeichert und dann nach SGRx gesprungen.

```
P23.5.1
SUMNEM  B40F 3A 33 12 FD 4E 00 FD 46 01 ED
        B419 43 2F 12 FD 46 02 FD 23 FD 23        0804
        B423 FD 23 FD BE 00 20 07 22 31 12
        B42D 2A 2F 12 E9 F5 7E FE 0D 23 20        0F80
        B437 FA F1 FD 23 10 E8 C9
SUMN02  B43E 3A 33 12 E6 CF FE 03 CA 12 B4        1911
        B448 FE 09 CA 12 B4 FE 0B CA 12 B4
        B452 C9                                   1F0A
SUMN2A  B453 3A 33 12 E6 CF FE C1 CA 12 B4
        B45D FE C5 CA 12 B4 C9                    28A9
SUMN03  B463 3A 33 12 E6 C7 FE 04 CA 12 B4
        B46D FE 05 CA 12 B4 FE 46 CA 12 B4        32CE
        B477 C9
SUMN3A  B478 3A 33 12 E6 F8 FE 70 CA 12 B4
        B482 E6 C0 FE 80 C0 3A 33 12 E6 F8        3F33
        B48C C3 12 B4
SUMN04  B48F 3A 33 12 E6 C0 FE 40 CA 12 B4        45AF
        B499 C9
```

```
SUMN05 B49A 3A 33 12 E6 C7 FE C0 CA 12 B4       4BF2
       B4A4 C9
SUMN5A B4A5 3A 33 12 E6 C7 FE C7 CA 12 B4       523C
       B4AF C9
SUMN08 B4B0 3A 33 12 E6 C7 FE 06 CA 12 B4       57C5
       B4BA C9
SUMN10 B4BB 3A 33 12 E6 CF FE 01 CA 12 B4       5D51
       B4C5 C9
SUMN11 B4C6 3A 33 12 E6 C7 FE C2 CA 12 B4       6396
       B4D0 FE C4 CA 12 B4 C9
SUMNC2 B4D6 3A 33 12 E6 F8 FE 30 C8 E6 C0       6DAA
       B4E0 FE 00 C0 3A 33 12 E6 F8 C3 12
       B4EA B4                                  734E
SUMNC3 B4EB 3A 33 12 E6 C7 FE 46 CA 12 B4
       B4F5 FE 86 CA 12 B4 FE C6 CA 12 B4       7EB6
       B4FF C9
SUMNC4 B500 3A 33 12 E6 C0 FE 40 CA 12 B4       8472
       B50A FE 80 CA 12 B4 FE C0 CA 12 B4
       B514 C9                                  8B97
SMNE2  B515 3A 33 12 E6 C7 FE 40 CA 12 B4
       B51F FE 41 CA 12 B4 C9                   9429
SUMNE3 B525 3A 33 12 E6 CF FE 42 CA 12 B4
       B52F FE 4A CA 12 B4 C9                   9CCE
SUMNE4 B535 3A 33 12 E6 CF FE 43 CA 12 B4
       B53F FE 4B CA 12 B4 C9                   A575
SUMND2 B545 3A 33 12 E6 CF FE 09 CA 12 B4
       B54F C9                                  AB09
SUMND4 B550 3A 33 12 E6 C7 FE 46 CA 12 B4
       B55A C9                                  B0D2
SUMD4A B55B 3A 33 12 E6 F8 FE 70 CA 12 B4
       B565 C9                                  B6F6
```

23.6 Die Übertragung in den Ausgabebereich

Die Subroutine BLTRAS transferiert den gesuchten String aus UGRx in den Ausgabebereich:

```
P23.6.1
B566 111212    BLTRAS: LD    DE,ABER+12H
B569 2A3112            LD    HL,(MNEMAD)
B56C 7E        LBT0:   LD    A,(HL)
B56D 12                LD    (DE),A
B56E 13                INC   DE
```

```
B56F 23                  INC  HL
B570 FE0D                CP   0DH
B572 20F8                JR   NZ,LBT0
B574 C9                  RET                   ;0454
```

Im folgenden werden die benötigten Opcodes in den Ausgabebereich übertragen und der Programmzähler IX nebst Pointer «VON» entsprechend der Länge des disassemblierten Befehls erhöht.

```
P23.6.2
INKR4   B575 CD AD B5 18 0F
INKR3   B57A CD B6 B5 18 0C                          04B2
INKR2   B57F CD BF B5 18 09
INKR1   B584 CD C8 B5 18 06 DD 23 DD 23 DD           0C59
        B58E 23 DD 23 DD 22 28 12 11 00 12
        B598 CD 15 00 CD 06 00 3A 2C 12 A7           11AC
        B5A2 33 33 CA 12 B0 CD A5 01 C3 12
        B5AC B0                                      1696
BYTE4   B5AD DD 56 03 21 0E 12 CD 74 B0 DD
        B5B7 56 02 21 0B 12 CD 74 B0 DD 56           1E95
        B5C1 01 21 08 12 CD 74 B0 DD 56 00
        B5CB 21 05 12 CD 74 B0 C9                    24E7
```

23.7 Ersatz von Platzhaltersymbolen

Die Routine ERS542 setzt den durch Bit5 bis Bit4 bestimmten 2stelligen Substituenten für das Platzhaltersymbol in den Ausgabebereich ein. Für die anderen SGR...-Routinen gilt das Entsprechende.

```
P23.7.1
ERS542  B5D2 CD E2 B6 3A 33 12 E6 30 CB 3F
        B5DC CB 3F CB 3F 32 E8 B5 32 EF B5           0ABD
        B5E6 FD 7E 00 77 FD 23 23 FD 7E 00
        B5F0 77 C9                                   10AD
ERS531  B5F2 CD E2 B6 3A 33 12 E6 38 CB 3F
        B5FC CB 3F CB 3F 32 05 B6 FD 7E 00           1A35
        B606 77 C9
ERS532  B608 CD E2 B6 3A 33 12 E6 38 CB 3F           2081
        B612 CB 3F 32 1C B6 32 23 B6 FD 7E
        B61C 00 77 FD 23 23 FD 7E 00 77 C9           298A
ERS201  B626 CD E2 B6 3A 33 12 E6 07 32 33
        B630 B6 FD 7E 00 77 C9
ERSnn   B636 CD E2 B6 CD 74 B0 C9                    3650
```

Die Routinen ERSxxx werden aufgerufen z. B. von EXb53. Dieses als Beispiel dargestellte Unterprogramm bewirkt den Austausch des Symbols «b» gegen den von Bit5 bis Bit3 bestimmten String. Dieser wird in Tb gefunden.

```
P23.7.2
EXb53   B63D FD 21 6E BE 3E 9A 32 34 12 CD
        B647 F2 B5 C9                          06D7
EXcc53  B64A FD 21 76 BE 3E 9F 32 34 12 CD
        B654 08 B6 C9                          0CD2
EXpp54  B657 FD 21 86 BE 3E 9E 32 34 12 CD
        B661 D2 B5 C9                          13A5
EXqq54  B664 FD 21 8E BE 3E A0 32 34 12 CD
        B66E D2 B5 C9                          1A82
EXr53   B671 FD 21 96 BE 3E 9D 32 34 12 CD
        B67B F2 B5 C9                          2184
EXr20   B67E FD 21 96 BE 3E 9D 32 34 12 CD
        B688 26 B6 C9                          27BB
EXss54  B68B FD 21 9E BE 3E A4 32 34 12 CD
        B695 D2 B5 C9                          2EAC
EXtt53  B698 FD 21 A6 BE 3E 96 32 34 12 CD
        B6A2 08 B6 C9                          34CE
```

EXPLMI ersetzt «+» durch «−», wenn dd > 7F.
EXXYXY ersetzt bis zu zweimal «X» durch «Y».
OFFS: Glücklicherweise stehen die Offsets zu den Indexregistern stets im dritten Byte.
SUCH sucht 22 Stellen des Ausgabebereichs nach dem in 1234 stehenden Symbol ab und übergibt die Adresse des ersten gesuchten Symbols an das HL-Register. Der Leser studiere im Z80-Handbuch aufmerksam die Erklärung des Blocksuchbefehls CPIR.

```
P23.7.3
EXPLMI  B6A5 F5 3E 2B 32 34 12 CD E2 B6 34
        B6AF 34 F1 C9                          065D
EXXYXY  B6B2 3A 35 12 A7 C8 CD BE B6 CD BE
        B6BC B6 C9 3E 58 32 34 12 CD E2 B6     110B
        B6C6 C0 34 C9
OFFS    B6C9 DD 7E 02 A7 F2 D5 B6 CD A5 B6     1971
        B6D3 ED 44 57 3E 9C 32 34 12 CD E2
        B6DD B6 CD 74 B0 C9                    216A
SUCH    B6E2 01 16 00 21 12 12 3A 34 12 ED
        B6EC B1 2B C9                          24D8
```

23.8 Fehlermeldung

Erreicht der Disassembler ein Byte, dessen Position keine Befehlsdekodierung zuläßt, wird eine Fehlermeldung ausgegeben und IX um eins erhöht. Das undeutbare Byte wird an den Ausgabebereich angefügt, so daß neben der Fehlermeldung z. B. für A3 ein «w» ausgegeben werden würde. Wenn in das Programm codierte Texte eingebettet sind, wird das die Deutung erleichtern. Je nach verwendetem Druckerinterface können von hier aus aber auch überraschende Druckerreaktionen ausgelöst werden. Man kann das durch NOPs in B702...B704 abstellen.

```
P23.8.1   (Prüfsumme: 0A20)

B6EF 3B           FEHLER: DEC  SP
B6F0 3B                   DEC  SP
B6F1 2116B7               LD   HL,TFEHL
B6F4 223112               LD   (MNEMAD),HL
B6F7 CD66B5               CALL BLTRAS
B6FA DD7E00               LD   A,(IX+00)
B6FD FE20                 CP   20H
B6FF DA05B7               JP   C,LFEH0
B702 322012               LD   (ABER+20H),A
B705 C384B5      LFEH0:   JP   INKR1

Prüfsumme (B000 ... B707) = 3A138
```

24
Fünf Spiele und ein Rätsel

Das vorliegende Buch soll eine Einführung in die Programmierung des Z80 sein und keine Programmsammlung. Dementsprechend lassen die folgenden Beispiele Algorithmen, Strategien und wenn möglich auch noch besondere Programmierkniffe erkennen.

24.1 Master Mind

Wir spielen nach folgenden Regeln: Die Grundmenge besteht aus den Zahlen 1, 2, ... 7. Der Computer «merkt» sich vier dieser Zahlen, und Sie sollen erraten, besser: «logisch erschließen», welche vier Zahlen das sind und in welcher Reihenfolge sie stehen. Sie werden aufgefordert, Ihre Vermutung zu äußern. Nehmen wir an, der Computer habe sich 1, 7, 4, 2 gemerkt, und Sie vermuten 2, 7, 5, 6. Dann sehen Sie auf dem Bildschirm

Ihr Zug: 2756 ri: 2 ri+Pos.: 1

was bedeutet: «Sie haben zwei Ziffern richtig vermutet (nämlich 2 und 7); eine davon steht zudem an der richtigen Stelle». Nun können Sie aufgrund dieser Zwischenmeldung weiterraten. Sie haben sieben Versuche, werden aber sicherlich diese Zahl nach Ihrem Geschmack variieren können.

Zunächst die Speicherdisposition (Arbeitsbereich):

1201 ... 1207	Hier stehen die Hexcodes der Grundmenge, und hier werden sie permutiert.
1208	Hier werden die Schleifen eines Spiels gezählt.
1209 ... 120C	Hexcodes der vier eingegebenen Zahlen.
120D	Anzahl richtig+Position
120E	Anzahl richtig geratene Zahlen.

Wir besprechen das Programm wieder in Abschnitten:
VORL setzt interne Abläufe auf Normalgeschwindigkeit zurück. Sie wurden von dem vom Verfasser verwendeten Diskettensystem verstellt.
MAMI ist das Hauptprogramm..
INIT initialisiert den Arbeitsbereich.
NEUSP steuert den Anfang einer neuen Spielrunde.
TAU35 bewirkt 35mal den Austausch von je zwei Elementen der Grundmenge. Als Randomgenerator wird dabei das Refresh-Register verwendet, was eine sorgfältige Abstimmung der Parameter erforderte, um störende Synchronmen zu vermeiden (vgl. Abschnitt 16.8).

```
P24.1.1
VORL    3FFB 3E 04 32 9E 11
MAMI    4000 CD 14 40 CD 21 40 CD 6F 40 CD      05BB
        400A C4 40 CD 02 41 CD 22 41 18 F2
INIT    4014 3E 31 06 07 21 01 12 77 3C 23      0B8F
        401E 10 FB C9
NEUSP   4021 11 8F 41 CD 15 00 CD 06 00 3E      1037
        402B 17 21 00 D8 06 0B 77 23 10 FC
        4035 AF 32 08 12                        13F9
TAU35   4039 DD 21 00 12 06 23 CD 60 40 32
        4043 53 40 32 5C 40 CD 60 40 32 56      1A27
        404D 40 32 59 40 DD 56 00 DD 5E 00
        4057 DD 72 00 DD 73 00 10 E0 C9         21F8
RND     4060 C5 06 93 CB 01 10 FC ED 5F E6
        406A 07 28 F4 C1 C9                     2A0D
```

Von den weiteren Programmabschnitten verdient PRUEF eine besondere Erwähnung. Nach Maßgabe der Tabelle TVGL wird das 1. mit dem 2., das 1. mit dem 3. usw. Element verglichen und auf Gleichheit geprüft. Dabei erledigt diese Routine zweierlei: Wenn VARBEF (4100) = 1C = INC E, dann werden richtige Positionen mitgezählt, wenn NOP, dann nicht. Wenn eine «Vermutung» beurteilt werden soll, zeigt IY auf GRMENG-01 und IX auf EINGB4-01. Soll hingegen eine Eingabe auf doppelte Elemente geprüft werden, weisen beide Pointer auf EINGB4-01.

```
P24.1.2
406F CD0600    EINGAB: CALL LETNL
4072 11AC41            LD   DE,TIHRZ
4075 CD1500            CALL MSG
```

```
4078 210812              LD    HL,SCHLFZ
407B 34                  INC   (HL)
407C 0604                LD    B,04
407E 23        LEING0:   INC   HL
407F CD8C40              CALL  ASCIN
4082 77                  LD    (HL),A
4083 CD9840              CALL  EKONT
4086 10F6                DJNZ  LEING0
4088 CDB740              CALL  GLCH?
408B C9                  RET                    ;0A97
408C CDB309    ASCIN:    CALL  ??KEY
408F CDCE0B              CALL  ?DACN
4092 F5                  PUSH  AF
4093 CD1200              CALL  PRNT
4096 F1                  POP   AF
4097 C9                  RET                    ;1154
4098 FE31      EKONT:    CP    31H
409A FAA340              JP    M,EFEHL
409D FE38                CP    38H
409F F2A340              JP    P,EFEHL
40A2 C9                  RET                    ;1834
```


```
4098 FE31      EKONT:    CP    31H
409A FAA340              JP    M,EFEHL
409D FE38                CP    38H
409F F2A340              JP    P,EFEHL
40A2 C9                  RET                    ;1834
```


```
40A3 119C41    EFEHL:    LD    DE,TEFEHL
40A6 CD1500              CALL  MSG
40A9 0610                LD    B,10H
40AB CD3E00    LEFL0:    CALL  BELL
40AE 10FB                DJNZ  LEFL0
40B0 33                  INC   SP
40B1 33                  INC   SP
40B2 33                  INC   SP
40B3 33                  INC   SP
40B4 C30340              JP    MAMI+03          ;1E02
40B7 DD210812  GLCH?:    LD    IX,EINGB4-01
40BB CDC840              CALL  PRUEF+04
40BE 7A                  LD    A,D
40BF FE04                CP    04
40C1 C8                  RET   Z
40C2 18DF                JR    EFEHL            ;242A
40C4 DD210012  PRUEF:    LD    IX,GRMENG-01
40C8 FD210812            LD    IY,EINGB4-01
40CC 216F41              LD    HL,TVGL
40CF AF                  XOR   A
40D0 57                  LD    D,A
```

```
40D1 5F                    LD    E,A
40D2 320041                LD    (VARBEF),A
40D5 060C                  LD    B,0CH
40D7 CDED40    LPR0:       CALL  PRUEF2                  ;2B27
40DA 10FB                  DJNZ  LPR0
40DC 3E1C                  LD    A,1CH
40DE 320041                LD    (VARBEF),A
40E1 0604                  LD    B,04
40E3 CDED40    LPR1:       CALL  PRUEF2
40E6 10FB                  DJNZ  LPR1
40E8 ED530D12              LD    (RI&POS),DE
40EC C9                    RET                           ;3236
40ED 7E        PRUEF2:     LD    A,(HL)
40EE 32F940                LD    (LP21+02),A
40F1 23                    INC   HL
40F2 7E                    LD    A,(HL)
40F3 32FC40                LD    (LP22+02),A
40F6 23                    INC   HL
40F7 DD7E00    LP21:       LD    A,(IX+00)
40FA FDBE00    LP22:       CP    (IY+00)
40FD 2002                  JR    NZ,LP23
40FF 14                    INC   D
4100 1C        VARBEF:     INC   E
4101 C9        LP23:       RET                           ;3A82
```

Das übrige ist trivial, so daß ein Hexdump genügt:

```
P24.1.3
ANZEIG  4102 11 B6 41 CD 15 00 3A 0E 12 F6
        410C 30 CD 12 00 11 BD 41 CD 15 00     063A
        4116 3A 0D 12 F6 30 CD 12 00 CD 06
        4120 00 C9                             0A34
GEWINN  4122 3A 0D 12 FE 04 28 08 3A 08 12
        412C FE 07 28 0F C9                    0E18
SIEGM   4131 CD 06 00 11 C8 41 CD 15 00 CD
        413B 06 00 18 1E                       11F0
ZUG7:   413F CD 06 00 11 E8 41 CD 15 00 CD
        4149 06 00 11 FD 41 CD 15 00 06 04     17ED
        4153 21 01 12 7E CD 12 00 23 10 F9
NEU?    415D CD 06 00 11 14 42 CD 15 00 CD     1D93
        4167 B3 09 FE 0A CA 21 40 C7
TVGL    416F 01 02 01 03 01 04 02 01 02 03     215D
        4179 02 04 03 01 03 02 03 04 04 01
        4183 04 02 04 03 01 01 02 02 03 03     2191
        418D 04 04
```

```
TMAMI   418F 16 4D A1 A4 96 92 9D 20 4D A6        2619
        4199 B0 9C 0D
TEFEHL  419C 20 45 A6 B0 97 A1 9A 92 AA 92        2CCD
        41A6 98 B8 92 9D 21 0D
TIHRZ   41AC 49 98 9D 20 5A A5 97 3A 20 0D        3315
TRI     41B6 20 20 9D A6 3A 20 0D
TRIP    41BD 20 20 9D A6 2B 50 B7 A4 3A 20        38B2
        41C7 0D
TGRAT   41C8 47 9D A1 96 A5 B8 A6 92 9D 92        3E9E
        41D2 2C 20 53 A6 92 20 98 A1 9A 92
        41DC B0 20 97 92 A3 B7 B0 B0 92 B0        48EF
        41E6 21 0D
TZUG7   41E8 44 A1 A4 20 A3 A1 9D 92 B0 20        4E09
        41F2 37 20 56 92 9D A4 A5 9F 98 92
        41FC 0D                                   5304
TRICHT  41FD 52 A6 9F 98 96 A6 97 20 A3 BB
        4207 9D 92 20 97 92 A3 92 A4 92 B0        5E17
        4211 3A 20 0D
TNEUSP  4214 4E 92 A5 92 A4 20 53 9E A6 92
        421E B8 3F 20 0D                          64A6
```

Viel Spaß mit diesem Spiel!

24.2 Leben

Dieses Spiel ist in zweifacher Hinsicht kein Spiel:

1. Es gibt einen Spieler, aber keinen Gegenspieler, keinen Sieger und keinen Verlierer. Man kann lediglich Figuren entwerfen und sie nach gewissen mathematischen Gesetzen verändern lassen; insofern handelt es sich um ein Grafikspiel.
2. Hinter diesem Spiel verbirgt sich ein höchst aktuelles Existenzproblem der gesamten Menschheit: Ein Mensch allein hat auf Dauer gesehen nur ein begrenztes Leben; irgendwann wird er sterben. Zwei Menschen könnten den Bestand erhalten. Bei einer angemessenen Besiedelungsdichte könnte sogar eine Vermehrung erfolgen.

Wird die Bevölkerungsdichte zu groß, so wird die Umwelt zerstört und die Bevölkerung geht schließlich zurück.

Diese Grundgedanken reduzierte John Horton Conway um 1970 auf ein abstraktes mathematisches Modell: Die Individuen sind auf einem unendlichen Schachbrett angesiedelt, wobei jedes bis zu acht Nachbarn (auf benachbarten Feldern) haben kann.

Bei 0 oder 1 Nachbarn stirbt das Individuum an Vereinsamung.

Bei zwei Nachbarn bleibt der Status quo erhalten: Hat ein unbesetztes Feld zwei Nachbarn, bleibt es unbesetzt; hat ein besetztes Feld zwei Nachbarn, bleibt es besetzt.

Bei 3 Nachbarn ist das Optimum erreicht, und das Feld wird gegebenenfalls mit einem neuen Individuum besetzt. Von 4 Nachbarn an ist die Belastbarkeit der Umwelt überschritten. Ist ein solches Feld besetzt, stirbt sein Inhaber wegen Übervölkerung.

Wir spielen auf den 1000 Feldern des Bildschirms, genauer: auf den 918 Feldern D029 ... D3BE (BFELD). Wir machen diese Einschränkung, um im weiteren Verlauf nicht über den Bildschirmrand hinauszugreifen. Dabei stellen wir uns den Bildschirm als eine vertikale Röhre vor, die durch Verkleben der beiden seitlichen Ränder entstand. Als Symbol für ein Individuum wählen wir den Buchstaben «X» mit dem Bildschirmcode 18h.

Als zweiten Speicherbereich (NFELD), auf dem die Nachbarn eines jeden Feldes gezählt werden, benutzen wir A429 ... A7BE; somit lassen wir Platz für eventuelles Benutzen des EBS oder eines Disassemblers, z. B. zum Relokatieren zwecks Ausbau des Programms. Wir beabsichtigen folgende Bedienung des Programms:

Zunächst wird – z. B. durch Reset – der Bildschirm möglichst frei gemacht. Dann zeichnet man mittels X und Cursorbewegungstasten eine Figur, bewegt den Cursor ein paar Zeilen tiefer und ruft mit J4000 das Programm auf.

Während man unter dem Aspekt der Ablaufgeschwindigkeit Master Mind sehr wohl auch hätte in BASIC abfassen können, wäre das hier geradezu qualvoll: Das erste Leben-Programm, das der Verfasser vor Jahren schrieb, benötigte zum Ausrechnen des neuen Bildes 60 Sekunden und zu seiner Ausgabe weitere 20, also eine Wartezeit von 80 Sekunden je Generation. Das folgende Maschinenprogramm liefert in 1 s rund drei Bilder! Wer will, mag noch eine Verzögerung einbauen.

Von einer kurzen Subroutine abgesehen, besteht das Programm nur aus einer einzigen endlosen Schleife. Zunächst muß NFELD auf null gesetzt werden:

```
P24.2.1
NULLS   4000 21 29 A4 11 BE A7 36 00 CD 7D
        400A 40 20 F9                            053D
```

Dann wird vom ersten bis zum letzten Feld von BFELD nach dem Schema

```
IX-29  IX-28  IX-27
IX-01         IX+01
IX+27  IX+28  IX+29 (alles hexadezimal!)
```

die Anzahl der Nachbarn ermittelt. Für jeden Nachbarn wird das entsprechende Byte von NFELD inkrementiert.

```
P24.2.2
NACHB   400D 21 29 A4 DD 21 29 D0 11 BE A7
        4017 3E 18 DD BE D7 20 01 34 DD BE        0913
        4021 D8 20 01 34 DD BE D9 20 01 34
        402B DD BE FF 20 01 34 DD BE 01 20        11B4
        4035 01 34 DD BE 27 20 01 34 DD BE
        403F 28 20 01 34 DD BE 29 20 01 34
        4049 CD 7D 40 DD 23 20 C7                 1BA2
```

Bei diesem Programmteil sollten Sie einmal überlegen, ob man nicht die achtfache Fleißwiederholung – z. B. mit einer Tabelle wie TVGL in P24.1.x – straffen könnte. Bedenken Sie dabei aber auch, daß dieser Abschnitt je Generation 918mal durchlaufen wird. Wie groß wäre etwa die zu erwartende Verlangsamung?

Im folgenden werden je nach Anzahl der Nachbarn neue Individuen erzeugt oder vorhandene gelöscht. Zunächst wird durch Maskierung mit FC ermittelt, ob vier oder mehr Nachbarn vorhanden sind. Die anschließende Maskierung mit FE stellt dann fest, ob es 0, 1, 2 oder 3 Nachbarn gibt. Bei weniger als zwei Nachbarn geht es zurück nach LAB11, und es wird gelöscht. Wenn aber XOR 03 drei Nachbarn ermittelt, wird ein X gesetzt. Bei zwei Nachbarn geschieht nichts.

```
P24.2.3
SETZ    4050 21 29 A4 DD 21 29 D0 11 BE A7
        405A 7E E6 FC 28 06                       06E9
LAB11   405F AF DD 77 00 18 0F 7E E6 FE 28
        4069 F5 7E EE 03 20 05 3E 18 DD 77        0FD0
        4073 00 CD 7D 40 DD 23 20 DF 18 83
ENDE?   407D E5 ED 52 E1 23 C9                    17E5
```

Noch eine Bemerkung zur Subroutine ENDE?: Sie ermittelt, ob schon das ganze Feld durchlaufen wurde. Wenn das der Fall ist, wird Zero-Flag gesetzt und nach RET wenig später abgefragt.

Sie können das Spiel mit folgenden «klassischen» Figuren beginnen:

```
Ampel:                      Blinker:
                                        XX
            XXX                         X
                                          X
                                         XX

Segler:                     Pentadekathelon:
              X                 X   X
            X X              XX XXXX XX
             XX                 X   X
```

Bild 24.1 Einige klassische Figuren

Es fasziniert nicht nur Sie, daß diese Figuren zu LEBEN beginnen. Erfinden Sie weitere!

24.3 Schlingpflanzen

Die beiden Spieler, Mensch und Computer, «laufen» auf dem Bildschirm umeinander herum und ziehen lange Bänder hinter sich her. Beide Spieler dürfen weder gegeneinander noch gegen die ausgelegten Bänder stoßen. Für den Computer ist das kein Problem, denn Geschwindigkeit ist keine Hexerei, schon gar nicht bei einem Maschinenprogramm! An das Reaktionsvermögen des Menschen werden jedoch – variabel – erhebliche Anforderungen gestellt. Er steuert den Weg seines Symbols mit den Cursorbewegungstasten.

Das Programm ist als Improvisation entstanden und nicht modular aufgebaut. Sie könnten manches verbessern.

Nach den trivialen Anfangsjobs wird mit dem Symbol A3 (Bildschirmcode) der Rand markiert. So braucht nicht durch Auswertung der Position geprüft zu werden, ob ein Spieler den Rand überschreitet.

Danach werden die Startorte beider Spieler mit deren Symbolen markiert und die DELTA-Werte für einen Schritt nach unten, oben, rechts und links in den Arbeitsbereich eingespeichert.

```
P24.3.1
1FFB 3E04      VORL:   LD    A,04
1FFD 329E11            LD    (119EH),A          ;0123
2000 113321    START:  LD    DE,TSCHLI
2003 CD1500            CALL  MSG
2006 CD0600            CALL  LETNL
2009 114421            LD    DE,TSYMB
200C CD1500            CALL  MSG
200F AF                XOR   A
2010 57                LD    D,A
2011 5F                LD    E,A
2012 CD3300            CALL  TIMST
2015 CDB309            CALL  ??KEY
2018 320812            LD    (SYMBM),A
201B 3E16              LD    A,16H
201D CD1200            CALL  PRNT               ;0A02
2020 0628      RAND:   LD    B,28H
2022 DD2100D0          LD    IX,OLIECK
2026 FD21C0D3          LD    IY,ULIECK
202A DD3600A3  LR0:    LD    (IX+00),A3H
202E FD3600A3          LD    (IY+00),A3H
2032 DD23              INC   IX
2034 FD23              INC   IY
2036 10F2              DJNZ  LR0
2038 0619              LD    B,19H
203A DD2100D0          LD    IX,OLIECK
203E DD3600A3  LR1:    LD    (IX+00),A3H
2042 DD3627A3          LD    (IX+27H),A3H
2046 112800            LD    DE,0028H
2049 DD19              ADD   IX,DE
204B 10F1              DJNZ  LR1                ;1D0D
204D DD21FCD1  STMECO: LD    IX,STARTM
2051 3A0812            LD    A,(SYMBM)
2054 DD7700            LD    (IX+00),A
2057 FD21EAD1          LD    IY,STARTC
205B FD360043          LD    (IY+00),43H
205F 112800            LD    DE,0028H
2062 ED530012          LD    (DELTAU),DE
2066 11D8FF            LD    DE,FFD8H
2069 ED530212          LD    (DELTAO),DE
206D 110100            LD    DE,0001
2070 ED530412          LD    (DELTAR),DE
2074 11FFFF            LD    DE,FFFFH
2077 ED530612          LD    (DELTAL),DE
207B ED530912          LD    (AKTOFF),DE        ;30C0
```

In der Randomroutine wird durch geeignete Maskierung eine der Zahlen 0, 2, 4, 6 gewonnen und in LOOP+02 in das Programm eingesetzt.

In ALTCPS wird die alte Computerposition auf den Stapel gerettet, denn es könnte sein, daß die neu berechnete nicht zulässig ist. Dann wird geprüft, ob eins der vier Nachbarfelder frei ist. Wenn ja, dann weiter bei NEUCPS; wenn nein, dann Text «Ich kann nicht ziehen» und weiter bei NEUSP.

NEUCPS berechnet aus der derzeitigen Computerposition in IY und der momentanen Random-Schrittrichtung in DE eine neue Position. Ist dieses Feld frei, dann weiter bei FELDFR; anderenfalls alte Position vom Stapel holen und zurück nach RND.

Im Verlauf von FELDFR bestimmt eine Warteroutine das Spieltempo. Anschließend wird geprüft, ob der Mensch eine neue Richtung eingab.

```
P24.3.2
207F 3A05E0    RND:     LD    A,(E005H)
2082 E606               AND   06
2084 328920             LD    (AKDELT+02),A
2087 ED5B0012 AKDELT:   LD    DE,(1200H)
208B FDE5     ALTCPS:   PUSH  IY
208D FD7ED8             LD    A,(IY-28H)
2090 A7                 AND   A
2091 281A               JR    Z,NEUCPS
2093 FD7EFF             LD    A,(IY-01)
2096 A7                 AND   A
2097 2814               JR    Z,NEUCPS
2099 FD7E01             LD    A,(IY+01)
209C A7                 AND   A
209D 280E               JR    Z,NEUCPS
209F FD7E28             LD    A,(IY+28H)
20A2 A7                 AND   A
20A3 2808               JR    Z,NEUCPS
20A5 115121             LD    DE,TKANNI
20A8 CD1500             CALL  MSG
20AB 1848               JR    NEUSP             ;1353
20AD FD19     NEUCPS:   ADD   IY,DE
20AF FD7E00             LD    A,(IY+00)
20B2 A7                 AND   A
20B3 2804               JR    Z,FELDFR
20B5 FDE1               POP   IY
20B7 18C6               JR    RND
20B9 33       FELDFR:   INC   SP
20BA 33                 INC   SP
```

```
20BB FD360043              LD    (IY+00),43H
20BF 3E38                  LD    A,DELAY
20C1 CD1721                CALL  WARTE2
20C4 CD1B00                CALL  GETKY
20C7 A7                    AND   A
20C8 2810                  JR    Z,NSCHRM
20CA 3D                    DEC   A
20CB CB27                  SLA   A
20CD E606                  AND   06
20CF 32D420                LD    (LFFR0+02),A
20D2 ED5B0012 LFFR0:       LD    DE,(1200H)
20D6 ED530912              LD    (AKTOFF),DE          ;2487
```

Der Rest bedarf wohl keiner weiteren Erklärung:

```
P24.3.3
20DA ED5B0912 NSCHRM: LD    DE,(AKTOFF)
20DE DD19             ADD   IX,DE
20E0 DD7E00           LD    A,(IX+00)
20E3 A7               AND   A
20E4 2828             JR    Z,FLDFR2
20E6 116721           LD    DE,TKRACH
20E9 CD1500           CALL  MSG
20EC CD0600           CALL  LETNL
20EF 117821           LD    DE,TVERL
20F2 CD1500           CALL  MSG                     ;07A3
20F5 060A    NEUSP:   LD    B,0AH
20F7 CD3E00  LNSP0:   CALL  BELL
20FA 10FB             DJNZ  LNSP0
20FC CD0600           CALL  LETNL
20FF 118C21           LD    DE,TNEUSP
2102 CD1500           CALL  MSG
2105 CDB309           CALL  ??KEY
2108 FE0A             CP    0AH                     ;'J'
210A CA0020           JP    Z,START
210D C7               RST   00
210E 3A0812  FLDFR2:  LD    A,(SYMBM)
2111 DD7700           LD    (IX+00),A
2114 C38720           JP    AKDELT                  ;1472
Weiter als Hexdump:

WARTE2 2117 F5 F5 F5 F1 3D F5 20 FB F1 F1
       2121 3D F5 F5 20 F4 F1 F1 F1 3D F5      0F3F
       212B F5 F5 20 EB F1 F1 F1 C9
TSCHLI 2133 16 53 9F 98 B8 A6 B0 97 9E AA      1B5D
       213D B8 A1 B0 A2 92 B0 0D
```

```
TSYMB   2144 49 98 9D 20 53 BD B3 9A B7 B8        24C1
        214E 3A 20 0D
TKANNI  2151 49 9F 98 20 A9 A1 B0 B0 20 B0        2A42
        215B A6 9F 98 96 20 A2 A6 92 98 92
        2165 B0 0D                                3096
TKRACH  2167 45 A4 20 98 A1 96 20 97 92 A9
        2171 9D A1 9F 98 96 21 0D                 3899
TVERL   2178 53 A6 92 20 98 A1 9A 92 B0 20
        2182 AB 92 9D B8 B7 9D 92 B0 21 0D        42CF
TNEUSP  218C 4E 92 A5 92 A4 20 53 9E A6 92
        2196 B8 3F 20 28 4A 2F 4E 29 20 0D        4A2F
```

24.4 Mikromühle

An diesem Spiel sollen die Grundzüge einer etwas anspruchsvolleren Strategie aufgezeigt werden.

Die neun Felder des quadratischen Spielbretts (Bild 24.2) werden abwechselnd besetzt; der Mensch beginnt – der Computer zieht nach. Besetzte Felder dürfen nicht noch einmal besetzt werden, auch werden keine Steine weggenommen. Gewonnen hat, wer als erster drei Steine in gerader Linie plaziert: in einer Zeile, in einer Spalte oder in einer Diagonale.

Intern wird der Zustand der neun Spielfelder als Feldwert FW(i) dargestellt: FW(4) = 1 bedeutet, daß Feld 4 unbesetzt ist, FW(5) = 2 zeigt

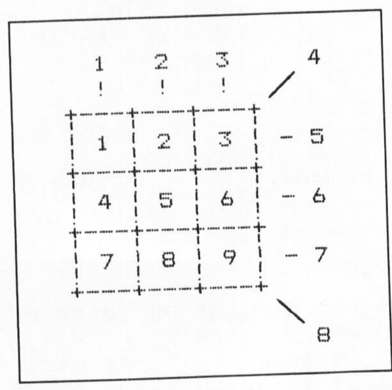


Bild 24.2 Mikromühle, Spielfeld und Numerierung der Mühlen

an, daß Feld 5 vom Menschen besetzt ist, und FW(6) = 3 verrät, daß Feld 6 vom Computer belegt wurde.

Nach jedem Zug werden die Mühlenprodukte MP(i) berechnet, die Produkte der zu einer Mühle gehörenden Feldwerte. Dabei können sich nur die Produkte 1, 2, 3, 4, 6, 8, 9, 12, 18, 27 ergeben, die Auskunft über die Gewinnchancen, aber auch Gefahren auf den acht Mühlen geben:

MP(i) = 4 besagt, daß zwei Felder der i-ten Mühle vom Menschen besetzt sind, während das dritte noch frei ist. Hier ist zu befürchten, daß der Mensch beim nächsten Zuge die Mühle schließt und damit gewinnt. Es besteht also eine hohe Dringlichkeit, das freie Feld zu besetzen.

MP(j) = 9 besagt, daß die entsprechende Chance für den Computer besteht. Natürlich soll er trotz der Gefahr auf Feld i den Sieg vorziehen.

Diese Überlegungen führen dazu, den acht Mühleprodukten acht Spielwerte SW(i) zuzuordnen, um dadurch schnell die größte Dringlichkeit zu finden:

MP =	1	2	3	4	5	6	7	12	18	27
SW =	4	3	6	7	2	9	8	1	1	10

Der Computer ermittelt nach jedem Zug für alle freien Felder das Maximum der Spielwerte der durch sie verlaufenden Mühlen und merkt sich zum momentanen Maximum den zugehörigen aktuellen Zug AZ, wobei ZX die Nummer des gerade geprüften Zuges ist. Der Suchlauf beginnt mit Feld 5, weil durch dieses die meisten Mühlen verlaufen.

Um Platz zu sparen, teilen wir das Programm in Disk-BASIC mit. Die vierstelligen Zahlen vor den Programmzeilen gehören natürlich nicht mit zum D-BASIC-Programm, sondern beziehen sich auf die Statements des Maschinenprogramms, das nur als Hexdump abgedruckt wird; Sie können es ja ohne Schwierigkeiten disassemblieren. – Das EBS wird zu diesem Programm nicht benötigt.

P24.4.1

```
100 REM ***************************************
101 REM **        M I K R O M U E H L E      **
102 REM ** MZ700D-BASIC V.04KP (DISK BASIC) **
103 REM ***************************************
104 DIM SAD(9),FW(9),MP(8),PS(27),SW(8):GOTO234
```

```
4000 105 GOSUB 113 :REM SPILF
4003 106 GOSUB 120 :REM INIT
4006 107 GOSUB 125 :REM ZUGM
4009 108 GOSUB 172 :REM VALZUG
400C 109 GOSUB 192 :REM ZUGC
400F 110 GOSUB 172 :REM VALZUG
4012 111 GOTO 107
     112 :
     113 REM SPIeLFeld zeichnen
401B 114 CLS
4020 115 PRINT T1$;: PRINT TT$: PRINT T2$
4032 116 PRINT T1$ : PRINT T3$: PRINT T1$
4044 117 PRINT T4$ : PRINT T1$: PRINT
4053 118 RETURN
     119 :
     120 REM INITiieren Anfangswerte
4054 121 FOR B=1 TO 9: FW(B)=1: NEXT B
4054 122 FOR B=1 TO 8: MP(B)=1: NEXT B
405E 123 RETURN
     124 :
     125 REM ZUGMensch
405F 126 PRINT IZ$
4065 127 GET Z$: IF Z$="" THEN 127
4068 128 IF ASC(Z$) < $31 THEN 125
406D 129 IF ASC(Z$) > $39 THEN 125
4072 130 PRINT Z$
407D 131 AZ=ASC(Z$)-$30
4082 132 GOSUB 137 :REM FFR?
40A6 133 IF FF=1 THEN 125
4085 134 FK=2: SY=$D: GOSUB 141 :REM SETZUG
4092 135 RETURN
     136 :
     137 REM FeldFRei?
4093 138 IF FW(AZ)=1 THEN FF=0: RETURN
40A0 139 PRINT SB$: FF=1: RETURN
     140 :
     141 REM SETZUG
40AA 142 FW(AZ)=FK
40BA 143 GOSUB 151: REM MUL
40BD 144 GOSUB 169: REM SETSAD
40C0 145 POKE ASAD,SY
40C7 146 RETURN
     147 :
     148 REM MUL2
40C8 149 MP(M)=MP(M)*FK: RETURN
     150 :
     151 REM MULtiplizieren
40D7 152 ON AZ GOTO 154, 156, 157, 159, 160, 162,
                                   163, 165, 166
40FF 153 END
```

```
4100 154 M=1: GOSUB 148: M=5: GOSUB 148
410C 155 M=8: GOSUB 148:                    RETURN
4113 156 M=2: GOSUB 148: M=5: GOSUB 148: RETURN
4120 157 M=3: GOSUB 148: M=4: GOSUB 148
412C 158 M=5: GOSUB 148:                    RETURN
4133 159 M=1: GOSUB 148: M=6: GOSUB 148: RETURN
4140 160 M=2: GOSUB 148: M=4: GOSUB 148
414C 161 M=6: GOSUB 148: M=8: GOSUB 148: RETURN
4159 162 M=3: GOSUB 148: M=6: GOSUB 148: RETURN
4166 163 M=1: GOSUB 148: M=4: GOSUB 148
4172 164 M=7: GOSUB 148:                    RETURN
4179 165 M=2: GOSUB 148: M=7: GOSUB 148: RETURN
4186 166 M=3: GOSUB 148: M=7: GOSUB 148
4192 167 M=8: GOSUB 148:                    RETURN
     168 :
     169 REM SETzeScreenADresse
4199 170 ASAD=SAD(AZ): RETURN
     171 :
     172 REM VALZUG von MP nach SW
41B5 173 FOR B=1 TO 8: SW(B)=PS(MP(B)): NEXT B
41CD 174 GOSUB 177: REM BEWERT
41D0 175 RETURN
     176 :
     177 REM BEWERTe den Spielstand
41D1 178 MW=1
41D6 179 FOR B=1 TO 8
41DB 180 IF SW(B)>MW THEN MW=SW(B)
41E6 181 NEXT B
41E8 182 IF MW<3 THEN PRINT UN$: GOTO 186
41EF 183 IF MW=9 THEN PRINT GR$: PRINT SG$: GOTO 186
41F3 184 IF MW=10THEN PRINT IG$: GOTO 186
41F7 185 RETURN
     186 REM NEUes SPiel?
4214 187 PRINT NS$;
421A 188 GET A$: IF A$="" THEN 188
421D 189 IF A$="J" THEN CLR: RUN
4221 190 END
     191 :
     192 REM ZUGComputer
4229 193 GOSUB 197: REM ZUGC?
422C 194 FK=3: SY=3: GOSUB 141: REM SETZUG
4237 195 RETURN
     196 :
     197 REM ZUGC?
4238 198 IF FW(5)=1 THEN AZ=5: RETURN
4245 199 MW=1          : REM MW = MAXSPW
424E 200 IF FW(1)>1 THEN 204
4255 201 ZX=1: PM=1: GOSUB 230: REM ZUGC2
425F 202 :      PM=2: GOSUB 230
4264 203 :      PM=8: GOSUB 230
4269 204 IF FW(2)>1 THEN 207
```

```
4270 205 ZX=2: PM=2: GOSUB 230
427A 206 :     PM=5: GOSUB 230
427F 207 IF FW(3)>1 THEN 211
4286 208 ZX=3: PM=3: GOSUB 230
4290 209 :     PM=4: GOSUB 230
4295 210 :     PM=5: GOSUB 230
429A 211 IF FW(4)>1 THEN 214
42A1 212 ZX=4: PM=1: GOSUB 230
42AB 213 :     PM=6: GOSUB 230
42B0 214 IF FW(6)>1 THEN 217
42B7 215 ZX=6: PM=3: GOSUB 230
42C1 216 :     PM=6: GOSUB 230
42C6 217 IF FW(7)>1 THEN 221
42CD 218 ZX=7: PM=1: GOSUB 230
42D7 219 :     PM=4: GOSUB 230
42DC 220 :     PM=7: GOSUB 230
42E1 221 IF FW(8)>1 THEN 224
42E8 222 ZX=8: PM=2: GOSUB 230
42F2 223 :     PM=7: GOSUB 230
42F7 224 IF FW(9)>1 THEN 228
42FE 225 ZX=9: PM=3: GOSUB 230
4308 226 :     PM=7: GOSUB 230
430D 227 :     PM=8: GOSUB 230
4312 228 RETURN
     229 :
     230 REM ZUGC2
4313 231 IF SW(PM)>MW THEN MW=SW(PM): AZ=ZX
432C 232 RETURN
     233 :
432D 234 TT$="  Mikromühle"
433A 235 T1$="+-+-+-+"
4342 236 T2$="!1!2!3!"
434A 237 T3$="!4!5!6!"
4352 238 T4$="!7!8!9!"
435A 239 IZ$="Ihr Zug? "
4364 240 SB$="schon besetzt!"
4374 241 UN$="Das war unentschieden."
438B 242 GR$="Gratuliere,"
4397 243 SG$="Sie haben gewonnen."
43AB 244 IG$="Ich habe gewonnen."
43BE 245 NS$="Neues Spiel? "
43CC 246 PS( 1)=4: PS( 2)= 3: PS(3)=6: PS( 4)=7
43D1 247 PS( 6)=2: PS( 8)= 9: PS(9)=8: PS(12)=1
43DD 248 PS(18)=1: PS(27)=10
43E7 249 SAD(1)=$D029: SAD(2)=$D02B: SAD(3)=$D02D
43ED 250 SAD(4)=$D079: SAD(5)=$D07B: SAD(6)=$D07D
43F3 251 SAD(7)=$D0C9: SAD(8)=$D0CB: SAD(9)=$D0CD
     252 GOTO 105
```

Und dazu der unstrukturierte Hexdump:

```
4000 CD 1B 40 CD 54 40 CD 5F 40 CD B5 41 CD 29 42 CD B5 41 18 F2
4014 CD 15 00 CD 06 00 C9 3E 16 CD 12 00 11 3A 43 CD 15 00 11 2D     ;0F1C
4028 43 CD 14 40 11 42 43 CD 14 40 11 3A 43 CD 14 40 11 4A 43 CD
403C 14 40 11 3A 43 CD 14 40 11 52 43 CD 14 40 11 3A 43 CD 14 40     ;1ACA
4050 CD 06 00 C9 06 11 21 00 12 36 01 23 10 FB C9 11 5A 43 CD 15
4064 00 CD B3 09 FE 21 FA 5F 40 FE 2A F2 5F 40 CD CE 0B F5 CD 12     ;2AE2
4078 00 CD 06 00 F1 E6 0F 32 1D 12 CD 93 40 3E 02 32 1B 12 3E 0D
408C 32 1C 12 CD AA 40 C9 DD 21 FF 11 32 9C 40 DD 7E 04 FE 01 C8     ;39A8
40A0 11 64 43 CD 14 40 33 33 18 B5 3A 1D 12 DD 21 FF 11 32 B9 40
40B4 3A 1B 12 DD 77 04 CD D7 40 CD 99 41 2A 1E 12 3A 1C 12 77 C9     ;47A2
40C8 46 48 CB 20 3A 1B 12 FE 02 78 28 01 81 77 C9 3A 1D 12 FE 01
40DC 28 22 FE 02 28 31 FE 03 28 3A FE 04 28 49 FE 05 28 52 FE 06     ;5546
40F0 28 67 FE 07 28 70 FE 08 28 7F FE 09 CA 86 41 76 21 09 12 CD
4104 C8 40 21 0D 12 CD C8 40 21 10 12 CD C8 40 C9 21 0A 12 CD C8     ;6506
4118 40 21 0D 12 CD C8 40 C9 21 0B 12 CD C8 40 21 0C 12 CD C8 40
412C 21 0D 12 CD C8 40 C9 21 09 12 CD C8 40 21 0E 12 CD C8 40 C9     ;7419
4140 21 0A 12 CD C8 40 21 0C 12 CD C8 40 21 0E 12 CD C8 40 21 10
4154 12 CD C8 40 C9 21 0B 12 CD C8 40 21 0E 12 CD C8 40 C9 21 09     ;8545
4168 12 CD C8 40 21 0C 12 CD C8 40 21 0F 12 CD C8 40 C9 21 0A 12
417C CD C8 40 21 0F 12 CD C8 40 C9 21 0B 12 CD C8 40 21 0F 12 CD     ;9141
4190 C8 40 21 10 12 CD C8 40 C9 3A 1D 12 3D CB 27 32 AB 41 32 B0
41A4 41 DD 21 E7 43 DD 6E 06 DD 23 DD 66 06 22 1E 12 C9 06 08 21     ;A00F
41B8 09 12 DD 21 CB 43 11 11 12 7E 32 C7 41 DD 7E 0C 12 23 13 10
41CC F4 CD D1 41 C9 3E 01 32 19 12 06 08 21 11 12 3A 19 12 BE 30     ;ABBE
41E0 04 7E 32 19 12 23 10 F3 3A 19 12 FE 03 38 09 FE 09 28 0D FE
41F4 0A 28 17 C9 11 74 43 CD 14 40 18 14 11 8B 43 CD 14 40 11 97     ;B773
4208 43 CD 14 40 18 06 11 AB 43 CD 14 40 11 BE 43 CD 15 00 CD B3
421C 09 FE 0A 28 01 C7 33 33 33 33 C3 00 40 CD 38 42 3E 03 32 1B     ;C42E
4230 12 32 1C 12 CD AA 40 C9 3A 04 12 FE 01 20 06 3E 05 32 1D 12
4244 C9 3E 01 32 19 12 DD 21 10 12 3A 00 12 FE 01 20 14 3E 01 32     ;CDAE
4258 1A 12 3E 01 CD 13 43 3E 05 CD 13 43 3E 08 CD 13 43 3A 01 12
426C FE 01 20 0F 3E 02 32 1A 12 3E 02 CD 13 43 3E 05 CD 13 43 3A     ;D727
4280 02 12 FE 01 20 14 3E 03 32 1A 12 3E 03 CD 13 43 3E 04 CD 13
4294 43 3E 05 CD 13 43 3A 03 12 FE 01 20 0F 3E 04 32 1A 12 3E 01     ;DF98
42A8 CD 13 43 3E 06 CD 13 43 3A 05 12 FE 01 20 0F 3E 06 32 1A 12
42BC 3E 03 CD 13 43 3E 06 CD 13 43 3A 06 12 FE 01 20 14 3E 07 32     ;E90A
```

```
42D0 1A 12 3E 01 CD 13 43 3E 04 CD 13 43 3E 07 CD 13 43 3A 07 12
42E4 FE 01 20 0F 3E 08 32 1A 12 3E 02 CD 13 43 3E 07 CD 13 43 3A   ;F2BF
42F8 08 12 FE 01 20 14 3E 09 32 1A 12 3E 03 CD 13 43 3E 07 CD 13
430C 43 3E 08 CD 13 43 C9 32 1E 43 32 22 43 3A 19 12 DD BE 06 D0   ;FD7F
4320 DD 7E 06 32 19 12 3A 1A 12 32 1D 12 C9 20 20 4D A6 A9 9D B7
4334 B3 AD 98 B8 92 0D 2B 2D 2B 2D 2B 2D 2B 0D 21 31 21 32 21 33   ;0985
4348 21 0D 21 34 21 35 21 36 21 0D 21 37 21 38 21 39 21 0D 49 98
435C 9D 20 5A A5 97 3F 20 0D A4 9F 98 B7 B0 20 9A 92 A4 92 96 A2   ;16B8
4370 96 21 20 0D 44 A1 A4 20 A3 A1 9D 20 A5 B0 92 B0 96 A4 9F 98
4384 A6 92 9C 92 B0 2E 0D 47 9D A1 96 A5 B8 A6 92 9D 92 2C 0D 53   ;2A0A
4398 A6 92 20 98 A1 9A 92 B0 20 97 92 A3 B7 B0 B0 92 B0 21 0D 49
43AC 9F 98 20 98 A1 9A 92 20 97 92 A3 B7 B0 B0 92 B0 2E 0D 4E 92   ;3E4F
43C0 A5 92 A4 20 53 9E A6 92 B8 3F 20 0D 04 03 06 07 00 02 00 09
43D4 08 00 00 01 00 00 00 00 00 01 00 00 00 00 00 00 00 00 0A 29   ;43F3
43E8 D0 2B D0 2D D0 79 D0 7B D0 7D D0 C9 D0 CB D0 CD D0            ;4F6D
```

24.5 Raummühle

Dieses schöne Spiel benutzt als Spielfeld einen Würfel, der in $4 \star 4 \star 4 = 64$ Teilwürfel unterteilt ist. Diese werden von Mensch und Computer abwechselnd besetzt mit dem Ziel, vier Steine in gerader Linie zu setzen, sei es von links nach rechts in einer Zeile, sei es von hinten-oben-links nach vorn-unten-rechts in einer Raumdiagonale usw. Die Strategie ist eine konsequente Fortführung von Spiel 24.2 und recht stark.

Es wird empfohlen, den Hexdump genau zu analysieren: Das Spiel wurde aufwendig ausgearbeitet und trickreich gestaltet:

1. Der Monitor wird aus dem ROM ins RAM verlagert und dort so verändert, daß das hier störende Scrolling auf den unteren Teil des Bildschirms begrenzt wurde.

2. Das Programm enthält einen Kopierschutz. Saven Sie es, bevor Sie es starten! Startadresse ist 4050. Zum Tüfteln können Sie (4036) = 11 durch C9 ersetzen...

```
4000 21 00 00 11 00 20 01 00 10 E5 C5 D5 ED B0 E1 C1 D1 D3 E0 ED
4014 B0 21 22 40 11 6D 0E 01 14 00 ED B0 18 14 01 6B 01 11 58 D2   ;00ED4
4028 21 80 D2 C5 ED B0 C1 D5 11 58 DA 21 80 DA 01 3E 00 21 3E 40
403C 11 3D 40 ED B8 21 3F 40 36 00 23 36 00 23 36 00 23 36 00 C9   ;01CB8
4050 CD 00 40 CD 6E 40 CD D7 40 CD F5 40 CD 18 43 CD D0 41 CD 18
4064 43 18 F2 CD 15 00 CD 06 00 C9 11 B7 43 CD 67 40 CD 06 00 11   ;02E3F
4078 D0 43 CD 67 40 11 F6 43 CD 67 40 3E 31 CD 12 00 11 6B 44 CD
408C 67 40 11 1D 44 CD 67 40 3E 32 CD 12 00 11 6B 44 CD 67 40 11   ;03C80
40A0 1D 44 CD 67 40 3E 33 CD 12 00 11 6B 44 CD 67 40 11 1D 44 CD
40B4 67 40 3E 34 CD 12 00 11 6B 44 CD 67 40 11 44 44 CD 67 40 11   ;04962
40C8 91 44 CD 67 40 CD 06 00 11 B7 44 CD 67 40 C9 06 8D 21 01 12
40DC 36 01 23 10 FB 06 40 11 4B 45 1A 6F 13 1A 67 13 36 00 CB DC   ;055E7
40F0 36 71 10 F2 C9 11 C5 44 CD 15 00 CD 3E 00 3E 02 32 1E 13 3E
4104 0D 32 1F 13 21 00 12 36 00 CD 6C 41 CD 6C 41 CD 6C 41 7E 32   ;06239
4118 20 13 CD 06 00 CD 21 41 C9 CD 51 41 3A 20 13 32 33 41 DD 21
412C 01 12 3A 1E 13 DD 77 01 2A 21 13 7E A7 20 0B 3A 1F 13 77 CD   ;06DD8
4140 84 43 CD 9F 41 C9 11 DB 44 CD 67 40 33 33 C3 F5 40 3A 20 13
4154 CB 27 32 62 41 32 67 41 DD 21 4B 45 DD 6E 04 DD 23 DD 66 04   ;07E49
4168 22 21 13 C9 CD B3 09 FE 21 FA 97 41 FE 25 F2 97 41 CD CE 0B
417C F5 CD 12 00 F1 3D CB 27 CB 27 CB 27 CB 27 CB 27 CB 27 CB 27   ;09215
4190 CB 16 CB 27 CB 16 C9 33 33 CD 06 00 C3 F5 40 11 CB 45 3A 20
41A4 13 6F 26 00 29 29 29 19 06 08 7E CD B6 41 23 10 F9 C9 E5 C5   ;0A169
41B8 11 41 12 6F 26 00 19 3A 1E 13 47 AF 86 10 FD 77 C1 E1 3E 24
41CC 32 8D 12 C9 3E 03 32 1F 13 32 1E 13 CD EB 41 CD 1C 42 CD 58   ;0AED5
41E0 42 CD BA 42 CD EA 42 CD 21 41 C9 DD 21 41 12 FD 21 8E 12 06
41F4 4C DD 7E 00 CD 09 42 FD 77 00 DD 23 FD 23 10 F1 FD 36 00 01   ;0C06E
4208 C9 C5 06 0F 21 FB 48 BE 28 06 23 23 10 F9 C1 C9 23 7E C1 C9
421C 21 CB 45 06 40 DD 21 DB 12 AF 32 1B 13 CD 37 42 3A 1B 13 DD   ;0D061
4230 77 00 DD 23 10 EF C9 C5 06 08 FD 21 8E 12 7E 32 4A 42 32 4F
4244 42 3A 1B 13 FD BE 4C 30 06 FD 7E 4C 32 1B 13 23 10 E8 C1 C9   ;0DFA1
4258 DD 21 DB 12 AF 32 1C 13 06 40 3A 1C 13 32 7B 42 32 84 42 AF
426C 32 23 13 CD 8F 42 3A 23 13 FE 02 38 0C DD 7E 3F FE 08 30 05   ;0EC70
4280 3E 08 DD 77 3F 3A 1C 13 3C 32 1C 13 10 D4 C9 C5 FD 21 41 12
4294 11 CB 45 3A 1C 13 6F 26 00 29 29 29 19 06 08 7E 32 A9 42 FD   ;0F88B
42A8 7E 4C FE 04 20 07 3A 23 13 3C 32 23 13 23 10 EB C1 C9 AF 32
42BC 1B 13 32 1C 13 21 DB 12 06 40 0E 00 DD 21 01 12 79 32 D2 42   ;103DC
42D0 DD 7E 3F FE 01 20 0E 3A 1B 13 BE 30 08 79 32 1C 13 7E 32 1B
```

```
42E4 13 0C 23 10 E3 C9 11 D0 44 CD 15 00 3A 1C 13 32 20 13 21 1C  ;10EB6
42F8 13 CB 06 CB 06 CD 0A 43 CD 0A 43 CD 0A 43 CD 06 00 C9 CB 06
430C CB 06 7E E6 03 3C F6 30 CD 12 00 C9 CD EB 41 AF 4F 32 1D 13  ;11EC1
4320 32 1B 13 06 4C 21 8E 12 3A 1B 13 BE 30 08 79 32 1D 13 7E 32
4334 1B 13 0C 23 10 EE 3A 1B 13 FE 03 38 09 FE 0B 28 10 FE 0C 28  ;12895
4348 14 C9 11 26 45 CD 67 40 CD 77 43 18 10 CD 9B 43 11 F3 44 18
435C F0 CD 9B 43 11 13 45 18 E8 11 3D 45 CD 67 40 CD B3 09 FE 0A  ;138B8
4370 C2 00 00 D1 C3 56 40 CD 3E 00 CD 3E 00 CD 3E 00 CD 3E 00 C9
4384 06 10 3E 66 CB DC AE 77 CB 9C C5 01 FF 4F 0B 78 A7 20 FB C1  ;14AA0
4398 10 EC C9 11 CB 47 3A 1D 13 6F 26 00 29 29 19 06 04 E5 7E CD
43AC 54 41 CB DC 36 17 E1 23 10 F3 C9 16 20 20 20 20 20 20 52 20  ;157CD
43C0 A1 20 A5 20 B3 20 B3 20 AD 20 98 20 B8 20 92 0D 20 20 20 31
43D4 2E 45 9A 92 B0 92 20 20 32 2E 45 9A 92 B0 92 20 20 33 2E 45  ;165A0
43E8 9A 92 B0 92 20 20 34 2E 45 9A 92 B0 92 0D 20 20 D0 E0 D2 E0
43FC D2 E0 D2 E0 CE D0 E0 D2 E0 D2 E0 D2 E0 CE D0 E0 D2 E0 D2 E0  ;17FEC
4410 D2 E0 CE D0 E0 D2 E0 D2 E0 D2 E0 CE 0D 20 20 CB E0 81 E0 81
4424 E0 81 E0 D3 CB E0 81 E0 81 E0 81 E0 D3 CB E0 81 E0 81 E0 81  ;19C7D
4438 E0 D3 CB E0 81 E0 81 E0 81 E0 D3 0D 20 20 CD E0 D1 E0 D1 E0
444C D1 E0 DD CD E0 D1 E0 D1 E0 D1 E0 DD CD E0 D1 E0 D1 E0 D1 E0  ;1BB18
4460 DD CD E0 D1 E0 D1 E0 D1 E0 DD 0D 20 FD 20 FD 20 FD 20 FD 20
4474 FD FD 20 FD 20 FD 20 FD 20 FD FD 20 FD 20 FD 20 FD 20 FD FD  ;1D50F
4488 20 FD 20 FD 20 FD 20 FD 0D 20 20 20 31 20 32 20 33 20 34 20
449C 20 31 20 32 20 33 20 34 20 20 31 20 32 20 33 20 34 20 20 31  ;1DE5F
44B0 20 32 20 33 20 34 0D 20 20 20 20 20 20 20 20 20 20 45 5A 53
44C4 0D 49 98 9D 20 5A A5 97 3A 20 20 0D 4D 92 A6 B0 20 5A A5 97  ;1E94A
44D8 3A 20 0D 46 92 B8 9C 20 A6 A4 96 20 A4 9F 98 B7 B0 20 9A 92
44EC A4 92 96 A2 96 21 0D 47 9D A1 96 A5 B8 A6 92 9D 92 2C 20 53  ;1FC3B
4500 A6 92 20 98 A1 9A 92 B0 20 97 92 A3 B7 B0 B0 92 B0 21 0D 49
4514 9F 98 20 98 A1 9A 92 20 97 92 A3 B7 B0 B0 92 B0 2E 0D 44 A1  ;21085
4528 A4 20 A6 A4 96 20 A5 B0 92 B0 96 A4 9F 98 A6 92 9C 92 B0 21
453C 0D 4E 92 A5 92 A4 20 53 9E A6 92 B8 3F 20 0D A3 D0 A5 D0 A7  ;2254C
4550 D0 A9 D0 F3 D0 F5 D0 F7 D0 F9 D0 43 D1 45 D1 47 D1 49 D1 93
4564 D1 95 D1 97 D1 99 D1 AC D0 AE D0 B0 D0 B2 D0 FC D0 FE D0 00  ;2423B
4578 D1 02 D1 4C D1 4E D1 50 D1 52 D1 9C D1 9E D1 A0 D1 A2 D1 B5
458C D0 B7 D0 B9 D0 BB D0 05 D1 07 D1 09 D1 0B D1 55 D1 57 D1 59  ;25A4A
45A0 D1 5B D1 A5 D1 A7 D1 A9 D1 AB D1 BE D0 C0 D0 C2 D0 C4 D0 0E
45B4 D1 10 D1 12 D1 14 D1 5E D1 60 D1 62 D1 64 D1 AE D1 B0 D1 B2  ;27471
```

```
45C8 D1 B4 D1 00 10 20 30 38 40 48 4C 01 10 21 39 4C 4C 4C 4C 02
45DC 10 22 3A 4C 4C 4C 4C 03 10 23 34 3B 44 4A 4C 04 14 20 31 4C   ;27DA0
45F0 4C 4C 4C 05 14 21 40 4C 4C 4C 4C 06 14 22 44 4C 4C 4C 4C 07
4604 14 23 35 4C 4C 4C 4C 08 18 20 32 4C 4C 4C 4C 09 18 21 44 4C   ;285F5
4618 4C 4C 4C 0A 18 22 40 4C 4C 4C 4C 0B 18 23 36 4C 4C 4C 4C 0C
462C 1C 20 33 3C 44 49 4C 0D 1C 21 3D 4C 4C 4C 4C 0E 1C 22 3E 4C   ;28E56
4640 4C 4C 4C 0F 1C 23 37 3F 40 4B 4C 00 11 24 41 4C 4C 4C 4C 01
4654 11 25 30 4C 4C 4C 4C 02 11 26 34 4C 4C 4C 4C 03 11 27 45 4C   ;2967B
4668 4C 4C 4C 04 15 24 38 4C 4C 4C 4C 05 15 25 31 39 41 48 4C 06
467C 15 26 35 3A 45 4A 4C 07 15 27 3B 4C 4C 4C 4C 08 19 24 3C 4C   ;29E88
4690 4C 4C 4C 09 19 25 32 3D 45 49 4C 0A 19 26 36 3E 41 4B 4C 0B
46A4 19 27 3F 4C 4C 4C 4C 0C 1D 24 45 4C 4C 4C 4C 0D 1D 25 33 4C   ;2A6DB
46B8 4C 4C 4C 0E 1D 26 37 4C 4C 4C 4C 0F 1D 27 41 4C 4C 4C 4C 00
46CC 12 28 42 4C 4C 4C 4C 01 12 29 34 4C 4C 4C 4C 02 12 2A 30 4C   ;2AF41
46E0 4C 4C 4C 03 12 2B 46 4C 4C 4C 4C 04 16 28 3C 4C 4C 4C 4C 05
46F4 16 29 35 3D 42 4B 4C 06 16 2A 31 3E 46 49 4C 07 16 2B 3F 4C   ;2B77B
4708 4C 4C 4C 08 1A 28 38 4C 4C 4C 4C 09 1A 29 36 39 46 4A 4C 0A
471C 1A 2A 32 3A 42 48 4C 0B 1A 2B 3B 4C 4C 4C 4C 0C 1E 28 46 4C   ;2BFD7
4730 4C 4C 4C 0D 1E 29 37 4C 4C 4C 4C 0E 1E 2A 33 4C 4C 4C 4C 0F
4744 1E 2B 42 4C 4C 4C 4C 00 13 2C 34 3C 43 4B 4C 01 13 2D 3D 4C   ;2C84C
4758 4C 4C 4C 02 13 2E 3E 4C 4C 4C 4C 03 13 2F 30 3F 47 49 4C 04
476C 17 2C 35 4C 4C 4C 4C 05 17 2D 43 4C 4C 4C 4C 06 17 2E 47 4C   ;2D0B7
4780 4C 4C 4C 07 17 2F 31 4C 4C 4C 4C 08 1B 2C 36 4C 4C 4C 4C 09
4794 1B 2D 47 4C 4C 4C 4C 0A 1B 2E 43 4C 4C 4C 4C 0B 1B 2F 32 4C   ;2D95F
47A8 4C 4C 4C 0C 1F 2C 37 38 47 4A 4C 0D 1F 2D 39 4C 4C 4C 4C 0E
47BC 1F 2E 3A 4C 4C 4C 4C 0F 1F 2F 33 3B 43 48 4C 10 00 30 20 11   ;2E180
47D0 01 31 21 12 02 32 22 13 03 33 23 14 04 34 24 15 05 35 25 16
47E4 06 36 26 17 07 37 27 18 08 38 28 19 09 39 29 1A 0A 3A 2A 1B   ;2E626
47F8 0B 3B 2B 1C 0C 3C 2C 1D 0D 3D 2D 1E 0E 3E 2E 1F 0F 3F 2F 01
480C 00 03 02 11 10 13 12 21 20 23 22 31 30 33 32 05 04 07 06 15   ;2EAB2
4820 14 17 16 25 24 27 26 35 34 37 36 09 08 0B 0A 19 18 1B 1A 29
4834 28 2B 2A 39 38 3B 3A 0D 0C 0F 0E 1D 1C 1F 1E 2D 2C 2F 2E 3D   ;2F016
4848 3C 3F 3E 04 00 0C 08 05 01 0D 09 06 02 0E 0A 07 03 0F 0B 14
485C 10 1C 18 15 11 1D 19 16 12 1E 1A 17 13 1F 1B 24 20 2C 28 25   ;2F37C
4870 21 2D 29 26 22 2E 2A 27 23 2F 2B 34 30 3C 38 35 31 3D 39 36
4884 32 3E 3A 37 33 3F 3B 11 00 33 22 15 04 37 26 19 08 3B 2A 1D   ;2FA2E
4898 0C 3F 2E 12 03 30 21 16 07 34 25 1A 0B 38 29 1E 0F 3C 2D 14
```

```
48AC 00 3C 28 15 01 3D 29 16 02 3E 2A 17 03 3F 2B 18 0C 30 24 19  ;2FF28
48C0 0D 31 25 1A 0E 32 26 1B 0F 33 27 05 00 0F 0A 15 10 1F 1A 25
48D4 20 2F 2A 35 30 3F 3A 09 0C 03 06 19 1C 13 16 29 2C 23 26 39  ;303DA
48E8 3C 33 36 15 00 3F 2A 19 0C 33 26 16 03 3C 29 1A 0F 30 25 01
48FC 03 02 04 03 05 04 06 06 02 08 09 09 07 0C 02 10 0B 12 02 18
4910 01 1B 0A 24 01 36 01 51 0C                                    ;307F0
```

24.6 Password

Oft sollen Programme (und Daten) gegen unbefugten Zugriff geschützt werden. Es gibt dafür mancherlei Techniken unterschiedlicher Wirksamkeit. Häufig wird zu Beginn nach einem Password (Schlüsselwort) gefragt. Gibt der Benutzer das richtige Wort ein, wird der Zugriff freigegeben. Anderenfalls können Programm und Daten gegen weitere Manipulation geschützt werden. Hierfür ein Beispiel, jedoch als Kopfzerbrecher verpackt.

```
P24.6.1
2000 11 50 20 CD 15 00 CD 06 00 11 6A 20
200C CD 15 00 11 8A 20 CD 03 00 3E 42 32   ;05F0
2018 65 20 06 04 DD 21 65 20 DD 7E 00 DD
2024 BE 27 20 17 DD 23 10 F4 3E A3 32 65   ;0ED2
2030 20 11 6D 20 CD 15 00 06 18 CD 3E 00
203C 10 FB 76 3E A3 32 65 20 11 7D 20 CD   ;162F
2048 15 00 3E C7 32 00 20 76 16 4B B0 A1
2054 9F A9 92 B0 20 53 A6 92 20 9C A1 A4   ;1FF9
2060 20 50 A1 A4 A4 A3 B7 9D 9C 0D 3F 20
206C 0D 49 9F 98 20 97 9D A1 96 A5 B8 A6   ;2B6C
2078 92 9D 92 21 0D 50 92 9F 98 20 97 92
2084 98 A1 9A 96 21 0D 0D 0D 0D 0D 0D 0D   ;33A2
```

25
Zum Abschluß

Sie haben nun die Grundzüge des Programmierens in Maschinensprache kennengelernt. Ein weites Feld tut sich damit vor Ihnen auf. Aber auch für die Arbeit mit BASIC bzw. dem BASIC-Interpreter eröffnen sich im Umgang mit einem Clean-Computer neue Möglichkeiten. Der Interpreter ist ja auch ein Maschinenprogramm, und beim *MZ-700* steht es im RAM und kann daher verändert werden. In dem Büchlein von BBG [4] finden Sie u. a. Möglichkeiten, den LIST-Befehl abzuändern. Mit der in diesem Buch gewonnenen Kenntnis über die Verschlüsselung von Texten müßte es Ihnen eigentlich möglich sein, per BASIC die «Error»-Meldungen in «Mist!» umzufunktionieren; zugegebenermaßen ein fragwürdiger Gag. Ernsthafter ist folgende Manipulation:

Das vorliegende Buch wurde umständehalber mit einem schnell zusammengeschriebenen BASIC-Textverarbeitungsprogramm zu Papier gebracht. Die Textzeilen wurden mit INPUT eingegeben. Bekanntlich dürfen dann aber keine Kommas im Text vorkommen, weil sonst das hinter ihnen Stehende ignoriert wird. Folgende Überlegung führte zur Abhilfe: Der BASIC-Interpreter erkennt das Komma vermutlich durch einen Vergleich CP 2C. Wo also folgen im BASIC-Interpreter die beiden Bytes FE und 2C aufeinander? Mit einem Testprogramm wurde dann probiert, an welcher Stelle das 2C durch den Code eines nicht benötigten Zeichens ersetzt werden muß, damit ein im Text stehendes Komma korrekt angenommen wird. Lösung: Vor dem INPUT POKE $2663,$BE und nach den INPUT POKE $2663,$2C. Beim benutzten D-BASIC ist die fragliche Adresse $28AD.

25.1 Die Ausgabe von BASIC-Zeilennummern

```
P25.1.1
240 LIMIT$CEFF
248 PRINT"AAA"
256 PRINT"BBBB"
264 PRINT"CCCCC"
272 PRINT"DDDDDD"
280 PRINT"EEEEEEE"
288 DATA $D3,$E4,$3E,$16,$CD,$12,$00,$FD,$21,$CF
296 DATA $6B,$FD,$6E,$00,$FD,$66,$01,$FD,$5E,$02
304 DATA $FD,$56,$03,$CD,$06,$00,$CD,$26,$CF,$CD
312 DATA $30,$CF,$E5,$C1,$FD,$09,$18,$E5,$E5,$01
320 DATA $00,$00,$A7,$ED,$42,$E1,$C0,$76,$7A,$CD
328 DATA $39,$CF,$7B,$CD,$39,$CF,$C9,$F5,$CB,$3F
336 DATA $CB,$3F,$CB,$3F,$CB,$3F,$CD,$DA,$03,$CD
344 DATA $12,$00,$F1,$CD,$DA,$03,$CD,$12,$00,$C9
352 FOR I=0 TO 79
360 READ A: POKE $CF00+I,A
368 NEXT I
376 USR($CF00)
384 REM Für D-BASIC $6BCF -> $8067
```

Dieses Programm gibt die verwendeten Zeilennummern aus. Versuchen Sie, die Wirkungsweise zu verstehen. Dabei wird sich Ihnen ein interessantes Knobelproblem auftun. Vielleicht gelingt es Ihnen, auch Programme zu erfassen, die bis in den Bereich oberhalb von CFFF hineinreichen? Die Startadresse 8067 gilt für das D-BASIC von Gischel; für das S-BASIC von Sharp gilt 6BCF.

Stört es Sie, daß die Zeilennummern hexadezimal ausgegeben werden? Nun, das müßten Sie ja jetzt ändern können!

Richtig interessant und nützlich wird das Programm aber erst dann, wenn es nur die Nummern der Zeilen ausgibt, in denen eine bestimmte, von Fall zu Fall frei bestimmbare Variable vorkommt. Das kann bei der Fehlersuche in sehr langen BASIC-Programmen recht nützlich sein.

25.2 Feingrafik mit BASIC

Man kann auf dem in Kapitel 19 begangenen Weg die Auflösung des Bildschirms vervierfachen. Hierzu folgendes Programm:

```
P25.2.1
100 LIMIT $CEFF: PI=4 * ATN(1)
110 DATA $D3,$E3,$3E,$F0,$32,$00,$D0,$21,$00,$D0
120 DATA $11,$01,$D0,$01,$E8,$03,$ED,$B0,$D3,$E1
130 DATA $C9,$F5,$C5,$D5,$E5,$21,$00,$00,$06,$00
140 DATA $CB,$3C,$CB,$10,$CB,$3D,$CB,$10,$CB,$00
150 DATA $CB,$00,$CB,$00,$3E,$C6,$80,$32,$3F,$CF
160 DATA $16,$00,$5C,$26,$D0,$06,$28,$19,$10,$FD
170 DATA $D3,$E3,$CB,$C6,$D3,$E1,$E1,$D1,$C1,$F1
180 DATA $C9
190 FOR I=0 TO 70: READ A: POKE $CF00+I,A: NEXT I
200 USR($CF00)
210 A0=24: GOSUB 300
220 A0=16: GOSUB 300
230 A0= 8: GOSUB 300
240 GOTO 240
250 :
300 FOR X=0 TO 79: Y=INT(25-A0*SIN(PI*X/24)-.5)
310 POKE $CF1A,X: POKE $CF1B,Y: USR($CF15): NEXT X
320 RETURN
```

Sicherlich fallen Ihnen viele weitere Anwendungsmöglichkeiten ein!

25.3 Hinweise zu RENUMBER

S-BASIC stellt schon den Befehl RENUM bereit. Dennoch sei die Frage erlaubt, wie denn so etwas wohl funktioniere.

Das völlig sinnlose BASIC-Programm P25.3.1 ist nicht zum Ablauf bestimmt; an ihm soll vielmehr etwas aufgezeigt werden.

```
P25.3.1  10 IF A=B THEN 20
         20 GOSUB 1000
         30 ON X GOTO 10, 20
         40 GOTO 20
         1000 PRINT A
         1010 RETURN
```

Es steht so im Speicher:

```
     Delim ZeiNr
6BCF 10 00 0A 00 93 20 41 F4 42 20 E2 20 0B 14 00 00
6BDF 0A 00 14 00 81 20 0B E8 03 00
6BE9 13 00 1E 00 9D 20 58 20 80 20 0B 0A 00 2C 20 0B
     14 00 00
6BFC 0A 00 28 00 80 20 0B 14 00 00
6C06 08 00 E8 03 8F 20 41 00
6C0E 06 00 F2 03 84 00
6C14 00 00
```

Die einzelnen Zeilen beginnen mit einem zwei Bytes langen Delimeter, der die Entfernung bis zum Beginn der nächsten Programmzeile angibt. Es folgen zwei Bytes mit der (natürlich ebenfalls) hexadezimal codierten Zeilennummer. Jede Zeile endet mit 00. Der Delimeter 0000 zeigt «Programmende» an.

93 ist das «Token» (Befehlscodierung) für "IF", E2 für "THEN", 81 für "GOSUB", 8F für "PRINT".

Wie also könnte RENUMBER aufgebaut sein?

Anhang

A Geschwindigkeit ist keine Hexerei

Das folgende BASIC-Programm wird ohne Erklärung der Arbeitsweise mitgeteilt. Tippen Sie es in Ihren Mz-700 ein:

```
100 CLS: LIMIT $AFFF: REM $AFFF = 45055
110 PRINT"Geschwindigkeit ist keine Hexerei!":PRINT
120 FOR I=45056 TO 45108: READ A
130 POKE I,A: NEXT I
140 PRINT"MENUE:"
150 PRINT" 1) BASIC      10mal"
160 PRINT" 2) MASCHINE 100mal"
170 CURSOR 0,6: INPUT"Kennzahl: ";K
180 ON K GOTO 210,260
190 PRINT"Fehlerhafte Eingabe!": END
200 :
210 FOR I=1 TO 10
220 FOR J=1 TO 26: POKE 54047+J,J: NEXT J
230 FOR J=1 TO 26: POKE 54047+J,0: NEXT J
240 NEXT I: USR($3E): GOTO 170
250 :
260 USR($B000): GOTO 170
270 :
280 DATA $F5,$C5,$D5,$E5,$06,$64,$F3,$D3,$E3,$CD
290 DATA $1C,$B0,$CD,$28,$B0,$D3,$E1,$FB,$10,$F2
300 DATA $CD,$3E,$00,$E1,$D1,$C1,$F1,$C9,$C5,$21
310 DATA $39,$D3,$06,$1A,$70,$2B,$10,$FC,$C1,$C9
320 DATA $C5,$21,$39,$D3,$06,$1A,$36,$00,$2B,$10
330 DATA $FB,$C1,$C9
```

Nach dem Programmstart mit RUN können Sie mit 1 oder 2 wählen, ob Sie das Alphabet von BASIC 10mal oder vom eingebundenen Maschinenprogramm 100mal dargeboten bekommen wollen. Urteilen Sie selbst!

Vielleicht möchten Sie, wenn Sie das Buch durchgearbeitet haben, auch dieses Maschinenprogramm verstehen.. Dazu das folgende Listing:

```
B000 F5      GkHex:  PUSH AF
B001 C5              PUSH BC
B002 D5              PUSH DE
B003 E5              PUSH HL
B004 0664            LD   B,64H        ;100mal
B006 F3      LOOP0:  DI                ;Interrupt gesp.
B007 D3E3            OUT  (E3H),A      ;VideoRAM ein
B009 CD1CB0          CALL INSCRN
B00C CD28B0          CALL CLSCRN
B00F D3E1            OUT  (E1H),A      ;Hauptspeicher
B011 FB              EI                ;Interrupt frei
B012 10F2            DJNZ LOOP0
B014 CD3E00          CALL BELL
B017 E1              POP  HL
B018 D1              POP  DE
B019 C1              POP  BC
B01A F1              POP  AF
B01B C9              RET               ;Ende Hauptprog.
B01C C5      INSCRN: PUSH BC
B01D 2139D3          LD   HL,ZEILE
B020 061A            LD   B,1AH        ;26 Buchstaben
B022 70      LOOP1:  LD   (HL),B
B023 2B              DEC  HL
B024 10FC            DJNZ LOOP1
B026 C1              POP BC
B027 C9              RET
B028 C5      CLSCRN: PUSH BC
B029 2139D3          LD   HL,ZEILE
B02C 061A            LD   B,1AH        ;26 Blanks
B02E 3600    LOOP2:  LD   (HL),00
B030 2B              DEC  HL
B031 10FB            DJNZ LOOP2
B033 C1              POP  BC
B034 C9              RET
```

Zu den beiden OUT-Befehlen können Sie auf den Seiten 94 und 127 des Sharp-Handbuchs nachlesen. DI und EI (disable bzw. enable interrupt) sperren alle «maskierbaren» Interrupts bzw. geben sie wieder frei. Interrupts sind im wesentlichen Steuersignale von peripheren Geräten, z. B. Drucker, Floppy, Bandlaufwerk.

Sehen Sie das Programm auch kritisch durch! Welches (andere) Befehlspaar ist mit Sicherheit überflüssig?

B Literaturverzeichnis

1. NN: *Z80-Assembler-Handbuch.* 1. Auflage. München: Ing. W. Hofacker GmbH, 1980.
2. ZAKS, R.: *Programmierung des Z80.* 4. Auflage. Düsseldorf: Sybex-Verlag, 1983.
3. NN: *MZ-700-Bedienerhandbuch.* Hamburg/Osaka: Sharp Corporation, 1983.
4. BIALKE, BERENDSEN, GLISZCZYNSKI: *Alles über den MZ-700.* 1. Auflage. Ahrensburg: BBG Software, 1983.
5. BAUMANN, R.: *Programmieren mit Pascal.* 1. Auflage. Würzburg: Vogel-Buchverlag, 1980.

Zu 1: Ein übersichtliches Handbuch. Der Leser sollte es (oder aber Nr. 2) anschaffen. Neun wesentliche Druckfehler: S. 129 lies 10001.r. statt 10000.r.; S. 233 lies 00ss1001 statt 01ss1010. Zu Seiten 169 bis 177: Bei allen ORs wird C-Flag gelöscht! S. 366: Vertausche «gerade» und «ungerade»; «M» bedeutet «Vorzeichen minus».

Zu 2: Viel systematische Information über die Programmierung des Z80. Auch Tabellen ähnlich 1.

Zu 3: Wer einen MZ-700 hat, besitzt auch dieses Handbuch. Wer mit einem anderen Z80-Computer arbeitet, täte gut daran, es sich zu beschaffen.

Zu 4: Zahlreiche nützliche Speicheradressen. Manipulation des BASIC-Interpreters. Auch Hinweise auf Maschinensprache mit Tabelle Mnemonics/Opcodes.

Zu 5: Hier lediglich Hinweis auf Erzeugung von Permutationen durch Rekursion.

C Die Lösungen der Aufgaben, Lösungsvorschläge

Die Lösungen der Aufgaben, Lösungsvorschläge

```
L3.1.1  a) 57  b) 41  c) 48  d) 65  e) 6C

L3.1.2                                            A  H  L
             Angenommener Anfangszust.:          12 34 56
             LD   A,H            7C              34 34 56
             LD   H,L            65              34 56 56
             LD   L,A            6F              34 56 34

L3.2.1  a) 87  b) 80  c) 81  d) 82

L3.2.2  2000  LD   A,(1200)     3A 00 12
        2003  LD   D,A          57
        2004  LD   A,(1201)     3A 01 12
        2007  ADD  A,D          82
        2008  LD   (1202),A     32 02 12
        200B  JP   0000         C3 00 00

L3.2.3  2000  3A 00 12 6F 3A 01 12 37
        2008  00 8D 32 02 12 C3 00 00

L3.3.1  2000  LD   A,(1202)     3A 02 12
        2003  LD   L,A          6F
        2004  LD   A,(120A)     3A 0A 12
        2007  ADD  A,L          85
        2008  LD   (1212),A     32 12 12
        200B  LD   A,(1201)     3A 01 12
        200E  LD   L,A          6F
        200F  LD   A,(1209)     3A 09 12
        2012  ADC  A,L          8D
        2013  LD   (1211),A     32 11 12
        2016  LD   A,(1200)     3A 00 12
        2019  LD   L,A          6F
        201A  LD   A,(1208)     3A 08 12
        201D  ADC  A,L          8D
        201E  LD   (1210),A     32 10 12
        2021  JP   0000         C3 00 00
```

L3.3.3 Die Zahl, von der subtrahiert wird, sollte oben stehen, der Subtrahend darunter. Wir müssen also zuerst den in der mittleren Zeile stehenden Subtrahenden ins H-Register laden und dann erst den in der oberen Zeile stehenden Minuenden in den Akkumulator:

```
        2000  LD   A,(120A)     3A 0A 12
        2003  LD   H,A          67
        2004  LD   A,(1202)     3A 02 12
        2007  SUB  A,H          94
        2008  DAA               27
```

```
        2009  LD   (1212),A     32 12 12
        200C  LD   A,(1209)     3A 09 12
        200F  LD   H,A          67
        2010  LD   A,(1201)     3A 01 12
        2013  SBC  A,H          9C
        2014  DAA               27
        2015  LD   (1211),A     32 11 12
        2018  LD   A,(1208)     3A 08 12
        201B  LD   H,A          67
        201C  LD   A,(1200)     3A 00 12
        201F  SBC  A,H          9C
        2020  DAA               27
        2021  LD   (1210),A     32 10 12
        2024  JP   0000         C3 00 00

L4.1.1  2000  LD   HL,ABCD      21 CD AB
    a)  2003  LD   A,H          7C
        2004  LD   (1200),A     32 00 12
        2007  LD   A,L          7D
        2008  LD   (1201),A     32 01 12
        200B  JP   0000         C3 00 00

    b)  200E  LD   HL,6789      21 89 67
        2011  LD   A,H          7C
        2012  LD   (1200),A     32 00 12
        2015  LD   A,L          7D
        2016  LD   (1201),A     32 01 12
        2019  JP   0000         C3 00 00

    c)  2011  JP   2003         C3 03 20

L4.2.2  2100  LD   DE,2301      11 01 23
    a)  2103  LD   (1200),DE    ED 53 00 12
        2107  LD   DE,6745      11 45 67
        210A  LD   (1202),DE    ED 53 02 12
        210E  JP   0000         C3 00 00

    b)  2200  LD   HL,2301      21 01 23
        2203  LD   (1200),HL    22 00 12
        2206  LD   HL,6745      21 45 67
        2209  LD   (1202),HL    22 02 12
        220C  JP   0000         C3 00 00

L4.3.2  2300  LD   HL,(1200)    2A 00 12
        2303  LD   (1202),HL    22 02 12
        2306  LD   HL,(1208)    2A 08 12
        2309  LD   (120A),HL    22 0A 12
        230C  LD   HL,(1201)    2A 01 12
        230F  LD   DE,(1209)    ED 5B 09 12
        2313  ADD  HL,DE        19
        2314  LD   (120F),HL    22 0F 12
        2317  LD   (1211),HL    22 11 12
        231A  JP   0000         C3 00 00
```

```
L4.3.3  ....  ...  ......       .. ..
        2007  SCF               37
        2008  CCF               3F
        2009  SBC  HL,DE        ED 52
        ....  ...  ......       .. ..
        Einen Befehl SUB HL,DE gibt es nicht.

L4.3.4  2400  LD   HL,(1200)    2A 00 12
        2403  ADD  HL,HL        29
        2404  ADD  HL,HL        29
        2405  ADD  HL,HL        29
        2406  LD   (1208),HL    22 08 12
        2409  JP   0000         C3 00 00

L4.4.1  2000  LD   A,00         3E 00
        2002  LD   HL,1200      21 00 12
        2005  LD   (HL),A       77
        2006  INC  HL           23
        2007  LD   (HL),A       77
        2008  INC  HL           23
        ::::  ::   ::::::       ::
        2012  INC  HL           23
        2013  LD   (HL),A       77
        2014  JP   0000         C3 00 00
Beurteilen Sie, ob bei diesem - zugegebenermaßen
unschönen - Programmaufbau der Befehl  LD (HL),00
vorzuziehen gewesen wäre.
Lösung: 8 * 2 > 8 * 1 + 2

L5.2.1  a) 7F (größter Vorwärtssprung)  b) 7D
        c) 67  d) EE  e) A1  f) zu weit
        g) viel zu weit; Rückwärtssprung!

L5.3.1  a) 200A  b) 200D  c) 210A - 200D = FD
        d) 200B  DJNZ 200A      10 FD

L5.4.1  2000  LD   A,(1200)     3A 00 12
        2003  CP   00           FE 00
        2005  JR   Z,2011       28 0A
        2007  LD   B,A          47
        2008  LD   A,(1201)     3A 01 12
        200B  LD   C,A          4F
        200C  LD   A,00         3E 00
        200E  ADD  A,C          81
        200F  DJNZ 200E         10 FD
        2011  LD   (1202),A     32 02 12
        2014  JP   0000         C3 00 00

L6.3.2  2100  LD   B,40         06 40
        2102  LD   A,FF         3E FF
        2104  LD   HL,1230      21 30 12
        2107  CALL AFFB         CD FB AF
        210A  JP   0000         C3 00 00
```

```
L7.1.2   3100  LD    HL,1200       21 00 12
         3103  LD    B,(HL)        46
         3104  INC   HL            23
         3105  LD    A,(HL)        7E
         3106  CALL  0012          CD 12 00
         3109  DJNZ  3104          10 F9
         310B  HALT                76

L7.2.1   3200  LD    HL,1200       21 00 12
         3203  LD    A,(HL)        7E
         3204  CP    2A            FE 2A
         3206  JR    Z,320E        28 06
         3208  CALL  0012          CD 12 00
         320B  INC   HL            23
         320C  JR    3203          18 F5
         320E  HALT                76

L7.2.2   3000  LD    A,(DE)        1A
         3001  CP    0D            FE 0D
         3003  JR    Z,300B        28 06
         3005  CALL  0012          CD 12 00
         3008  INC   DE            13
         3009  JR    3000          18 F5
         300B  RET                 C9

L7.3.3   1200                      56 4F 47 45 4C 2D
         1206                      42 55 43 48 56 45
         120C                      52 4C 41 47 0D
         4000  LD    DE,1200       11 00 12
         4003  LD    B,16          06 16
         4005  CALL  0015          CD 15 00
         4008  CALL  0006          CD 06 00
         400B  DJNZ  4005          10 F8
         400D  HALT                76

L10.2.1  1200                      12 34
         2000  LD    DE,(1200)     ED 5B 00 12
         2004  PUSH  DE            D5
         2005  POP   BC            C1
         2006  LD    (1202),BC     ED 43 02 12
         200A  JP    DUMP12        C3 DF AF

L10.2.2  1200                      12 34 56 78
         5000  LD    BC,(1200)     ED 4B 00 12
         5004  PUSH  BC            C5
         5005  LD    BC,(1202)     ED 4B 02 12
         5009  LD    (1200),BC     ED 43 00 12
         500D  POP   BC            C1
         500E  LD    (1202),BC     ED 43 02 12
         5012  JP    DUMP12        C3 DF AF
```

L10.3.4 Der Computer hängt. Beim CALL in Zeile 2028 wurde über dem Rücksprung nach 201E ein Rücksprung nach 202B vorgemerkt, und der wird nach dem Dekrementieren ausgeführt. Schematisch stellt sich das so dar:

```
                                     Stapel
                                   (unvollst.)
2028  CALL MSGNL
                                       ????
                             SP -> 202B
                                       201E
202B  DEC  SP
202C  DEC  SP
                             SP -> ????
                                       202B
                                       201E
202D  RET
                                       ????
                             SP -> 202B
                                       201E
202B  DEC  SP
....  ...  ..
```

```
L10.3.5 5100  PUSH HL          E5
        5101  PUSH HL          E5
        5102  INC  SP          33
        5103  POP  HL          E1
        5104  INC  SP          33
```

```
L10.3.7 2000  CALL 2025        CD 25 20  ;   '0.E'
        2003  CALL 200F        CD 0F 20  ;-> 1.E.
        2006  CALL 2025        CD 25 20  ;   '0.E'
        2009  CALL PAUSKY      CD D9 AF
        200C  JP   MONITR      C3 00 00
   1.E: 200F  CALL 2020        CD 20 20  ;   '1.E'
        2012  CALL 2019        CD 19 20  ;-> 2.E.
        2015  CALL 2020        CD 20 20  ;   '1.E'
        2018  RET              C9
   2.E: 2019  LD   DE,1210     11 10 12  ;   '2.E'
        201C  CALL MSGNL       CD ED AF
        201F  RET              C9
        2020  LD   DE,1208     11 08 12
        2023  JR   201C        18 F7
        2025  LD   DE,1200     11 00 12
        2028  JR   201C        18 F2
```

So wurden 2 Bytes eingespart. Gibt es wirksamere Verkürzungen?

```
L10.4.2 2000  LD   SP,2000     31 00 20
        2003  LD   HL,1234     21 34 12
        2006  LD   B,00        06 00
        2008  PUSH HL          E5
        2009  INC  HL          23
        200A  DJNZ 2008        10 FC
        200C  JP   DUMP12      C3 DF AF
```

L10.5.3

(A)	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0	FF
sec	.2	.5	1	1.5	2	3	4	5	7	10	13	16	20	25	30

L10.6.2 Ändern Sie (2001) zu 27 und (2011) zu 33. Überschreiben und ergänzen Sie dann das Programm so:

```
        201F  LD    B,27          06 27
        2021  LD    A,(IX+00)     DD 7E 00
        2024  LD    (IY+00),A     FD 77 00
        2027  LD    A,40          3E 40
        2029  CALL  WARTEN        CD B8 AF
        202C  LD    (IY+00),00    FD 36 00 00 ;space
        2030  DEC   IX            DD 2B
        2032  DEC   IY            FD 2B
        2034  DJNZ  2021          10 EB
        2036  JR    2000          18 C8
```

Vielleicht überlegen Sie noch, ob man die drei ersten Zeilen (2000 ... 2009) und den Rücksprung zwar nicht bytemäßig, aber doch gedanklich noch ein wenig straffen könnte.

L11.2.2 Die Binärdarstellung macht es deutlich:

```
  0001 0010   0011 0100   0101 0110   0111 1000
  0010 0100   0110 1000   1010 1100   1111 0000
```

```
L11.2.3 2020  LD    HL,1200       21 00 12
        2023  SRL   (HL)          CB 3E
        2025  INC   HL            23
        2026  RR    (HL)          CB 1E
        2028  INC   HL            23
        2029  RR    (HL)          CB 1E
        202B  INC   HL            23
        202C  RR    (HL)          CB 1E
        202E  JP    DUMP12        C3 DF AF
```

```
L11.3.2 2120  PUSH  HL            E5
        2121  PUSH  BC            C5
        2122  LD    HL,1200       21 00 12
        2125  LD    B,08          06 08
        2127  SCF                 37
        2128  CCF                 3F
        2129  RR    (HL)          CB 1E
        212B  INC   HL            23
        212C  DJNZ  2129          10 FB
        212E  POP   BC            C1
        212F  POP   HL            E1
        2130  RET                 C9
```

L11.4.4 Folgende PUSHs und POPs sind unnötig: 2100, 210F, 2120, 212F, 2140, 2155.

L12.1.2 Maskieren Sie zu a) mit 01, zu b) mit 03, 07, 0F, 1F, 3F, 7F und zu c) mit 80.

```
L12.2.1 2000  LD    A,(1200)      3A 00 12
        2003  AND   F0            E6 F0
        2005  LD    (1201),A      32 01 12
        2008  LD    A,(1200)      3A 00 12
        200B  AND   0F            E6 0F
        200D  LD    (1202),A      32 02 12
        2010  JP    DUMP12        C3 DF AF

L12.2.2 2000  LD    A,(1200)      3A 00 12
        2003  SRL   A             CB 3F
        2005  SRL   A             CB 3F
        2007  SRL   A             CB 3F
        2009  SRL   A             CB 3F
        200B  CALL  AUSGEB        CD 18 20
        200E  LD    A,(1200)      3A 00 12
        2011  CALL  AUSGEB        CD 18 20
        2014  CALL  PAUSWK        CD D4 AF
        2017  RST   00            C7
 AUSGEB 2018  AND   0F            E6 0F
        201A  OR    30            F6 30
        201C  CALL  PRNT          CD 12 00
        201F  RET                 C9

L12.2.3 Nur AUSGEB ist zu ändern:
 AUSHEX 2018  AND   0F            E6 0F
        201A  OR    30            F6 30
        201C  CP    3A            FE 3A
        201E  JR    C,2022        38 02
        2020  ADD   A,07          C6 07
        2022  CALL  PRNT          CD 12 00
        2025  RET                 C9

L13.1.1 2700  LD    A,(1201)      3A 01 12
        2703  LD    (1209),A      32 09 12
        2706  LD    A,(1200)      3A 00 12
        2709  LD    HL,1209       21 09 12
        270C  RLD                 ED 6F
        270E  LD    (1208),A      32 08 12
        2711  JP    DUMP12        C3 DF AF

L13.1.2 2100  PUSH  BC            C5
        2101  LD    HL,122F       21 2F 12
        2104  XOR   A             AF
        2105  LD    B,08          06 08
        2107  RLD                 ED 6F
        2109  DEC   HL            2B
        210A  DJNZ  2107          10 FB
        210C  POP   BC            C1
        210D  RET                 C9
```

Was bewirkt der Befehl XOR A ? - Lesen Sie unter 12.2 nach und berechnen Sie n XOR n. Beispiel:
Akkuinhalt als 1. Operand: 01101101
Akkuinhalt als 2. Operand: 01101101

```
L13.1.3 2120  PUSH BC           C5
        2121  LD   HL,1220      21 20 12
        2124  XOR  A            AF
        2125  LD   B,08         06 08
        2127  RRD               ED 67
        2129  INC  HL           23
        212A  DJNZ 2127         10 FB
        212C  POP  BC           C1
        212D  RET               C9

L14.2.3 3030  CALL NULL12       CD F4 AF
        3033  LD   HL,1200      21 00 12
        3036  LD   B,08         06 08
        3038  CALL ??KEY        CD B3 09
        303B  CALL ?DACN        CD CE 0B
        303E  LD   (HL),A       77
        303F  CALL PRNT         CD 12 00
        3042  INC  HL           23
        3043  DJNZ 3038         10 F3
        3045  JP   DUMP12       C3 DF AF

L14.5.1
IN6HEX: AF47  CALL INBYTE       CD 57 AF
        AF4A  LD   (HL),A       77
        AF4B  DEC  HL           2B
IN4HEX: AF4C  CALL INBYTE       CD 57 AF
        AF4F  LD   (HL),A       77
        AF50  DEC  HL           2B
IN2HEX: AF51  CALL INBYTE       CD 57 AF
        AF54  LD   (HL),A       77
        AF55  DEC  HL           2B
        AF56  RET               C9

L14.6.1
OUTBYT: AF30  PUSH AF           F5
        AF31  SRL  A            CB 3F
        AF33  SRL  A            CB 3F
        AF35  SRL  A            CB 3F
        AF37  SRL  A            CB 3F
        AF39  CALL ASC          CD DA 03
        AF3C  CALL PRNT         CD 12 00
        AF3F  POP  AF           F1
        AF40  CALL ASC          CD DA 03
        AF43  CALL PRNT         CD 12 00
        AF46  RET               C9
```

```
L14.6.2 2000  LD   HL,1200       21 00 12
        2003  LD   B,08          06 08
        2005  LD   A,(HL)        7E
        2006  CALL OUTBYT        CD 30 AF
        2009  INC  HL            23
        200A  DJNZ 2005          10 F9
        200C  CALL PAUSWK        CD D4 AF
        200F  RST  00            C7

L14.7.1
OUT6HX: AF20  LD   A,(HL)        7E
        AF21  CALL OUTBYT        CD 30 AF
        AF24  DEC  HL            2B
OUT4HX: AF25  LD   A,(HL)        7E
        AF26  CALL OUTBYT        CD 30 AF
        AF29  DEC  HL            2B
OUT2HX: AF2A  LD   A,(HL)        7E
        AF2B  CALL OUTBYT        CD 30 AF
        AF2E  DEC  HL            2B
        AF2F  RET                C9

L14.7.2 2900  LD   HL,130F       21 0F 13
        2903  LD   B,05          06 05
        2905  CALL PRINTS        CD 0C 00
        2908  CALL IN6HEX        CD 47 AF
        290B  DJNZ 2905          10 F8
        290D  CALL LETNL         CD 06 00
        2910  LD   HL,130F       21 0F 13
        2913  LD   B,05          06 05
        2915  CALL PRINTS        CD 0C 00
        2918  CALL OUT6HX        CD 20 AF
        291B  DJNZ 2915          10 F8
        291D  CALL LETNL         CD 06 00
        2920  JR   2900          18 DE
```

L14.8.2 a) Bit7 aus A wird über Carry zu Bit0 in B. b) Bit0 von B wird zu Bit3 von B. c) Das ehemalige Bit6 aus A wird ins Carry übertragen. d) Der Carry-Inhalt wird zu Bit0 von B und Bit3 von B wird zu Bit4 von B. e) Die Ausgabe erfolgt aus dem Akku.

L14.8.4 a) 301B b) 301B + E5 - 100 = 3000

```
L14.8.5
OUTBIN: AEF8  PUSH BC            C5
        AEF9  LD   C,A           4F
        AEFA  CALL BITBIT        CD 0B AF
        AEFD  CALL BITBIT        CD 0B AF
        AF00  CALL PRINTS        CD 0C 00
        AF03  CALL BITBIT        CD 0B AF
        AF06  CALL BITBIT        CD 0B AF
        AF09  POP  BC            C1
        AF0A  RET                C9
```

```
BITBIT: AF0B  LD    B,00         06 00
        AF0D  SLA   C            CB 21
        AF0F  RL    B            CB 10
        AF11  SLA   B            CB 20
        AF13  SLA   B            CB 20
        AF15  SLA   B            CB 20
        AF17  SLA   C            CB 21
        AF19  RL    B            CB 10
        AF1B  LD    A,B          78
        AF1C  CALL  OUTBYT       CD 30 AF
        AF1F  RET                C9
```

```
L14.8.6 2000  CALL  LETNL        CD 06 00
        2003  CALL  INBYTE       CD 57 AF
        2006  PUSH  AF           F5
        2007  CALL  PRINTS       CD 0C 00
        200A  POP   AF           F1
        200B  CALL  OUTBIN       CD F8 AE
        200E  JR    2000         18 F0
```

```
L15.1.5 7000  CALL  LETNL        CD 06 00
        7003  LD    HL,1201      21 01 12
        7006  CALL  IN4HEX       CD 4C AF
        7009  CALL  PRINTS       CD 0C 00
        700C  CALL  HEXDEZ       CD 00 20
        700F  LD    HL,1204      21 04 12
        7012  CALL  OUT6HX       CD 20 AF
        7015  JR    7000         18 E9
```

Die angegebenen Programmzeilen sind wie folgt zu ändern:

```
HEXDEZ   200D  RET               C9
SHILKS   2100  LD    HL,1200     21 00 12
         2105  INC   HL          23
ZFDOPC   2140  LD    HL,1202     21 02 12
SUBRUT   2151  INC   HL          23
```

L15.2.3 Disposition: Es sollen lediglich Zahlen bis zu 65535d = FFFFh ungewandelt werden.
1200 ... 1202: Fünfstellige Dezimalzahl (3 Bytes)
1206 ... 1207: Vierstellige Hexadezimalzahl
Stellenwerte: 10000d = 2710h, 1000d = 3E8h, 100d = 64h, 10d = Ah, 1d = 1h
1) Ergebnisfeld und IY auf Null setzen:

```
EFNULL   3100  XOR   A           AF
         3101  LD    B,1D        06 1D
         3103  LD    HL,1203     21 03 12
LAB1     3106  LD    (HL),A      77
         3107  INC   HL          23
         3108  DJNZ  LAB1        10 FC
         310A  LD    IY,0000     FD 21 00 00
         310E  RET               C9
```

```
2) Halbbytes-Swap der Dezimalzahl:
DZSWAP  3120  XOR  A            AF
        3121  LD   HL,1202      21 02 12
        3124  RLD               ED 6F
        3126  DEC  HL           2B
        3127  RLD               ED 6F
        3129  DEC  HL           2B
        312A  RLD               ED 6F
        312C  LD   B,A          47
        312D  RET               C9
3) Vor der Anzeige muß das Resultat in den Speicher:
VORANZ  3140  PUSH IY           FD E5
        3142  POP  DE           D1
        3143  LD   A,D          7A
        3144  LD   (1206),A     32 06 12
        3147  LD   A,E          7B
        3148  LD   (1207),A     32 07 12
        314B  RET               C9

4) Halbbytes-Swap und Addition IY = IY + DE
ADIYDE  3160  CALL DZSWAP       CD 20 31
        3163  AND  A            A7
        3164  RET  Z            C8
LADIYD  3165  ADD  IY,DE        FD 19
        3167  DJNZ LADIYD       10 FC
        3169  RET               C9
5) Zum Schluß das Hauptprogramm:
DEZHEX  3000  CALL EFNULL       CD 00 31
        3003  CALL DZSWAP       CD 20 31
        3006  LD   DE,2710      11 10 27
        3009  CALL ADIYDE       CD 60 31
        300C  LD   DE,03E8      11 E8 03
        300F  CALL ADIYDE       CD 60 31
        3012  LD   DE,0064      11 64 00
        3015  CALL ADIYDE       CD 60 31
        3018  LD   DE,000A      11 0A 00
        301B  CALL ADIYDE       CD 60 31
        301E  LD   DE,0001      11 01 00
        3021  CALL ADIYDE       CD 60 31
        3024  CALL VORANZ       CD 40 31
        3027  JP   DUMP12       C3 DF AF
```

L15.5.2 a) "Hexadezimal"; der Assembler verarbeitet auch Dezimalzahlen. b) 0006d = 0006h usw. c) DEFB d) DEFM (define message)

L16.1.5 a) (0000) = C3 b) (0000 ... 0100) = 63AC (0000 ... 0101) = 649C c) (0101) = 649C-63AC = F0 d) (0000 ... 0800) = 0328D0, (0800 ... 0FFF) = 0363C4, (0000 ... 0FFF) = 068C74
e) 0328D0 + 0363C4 = 068C94, 068C94 - 068C74 = 20

f) Vor dem Relokatieren macht man sich zweckmäßigerweise eine tabellarische Übersicht der maßgeblichen Adressen:

```
2000...202E  2100...2125  2140...2157  2800...2816
ADB0...ADDE  ADDF...AE04  AE05...AE1C  AE1D...AE33
PRUESU ADB0 CD 9F AF 11 1D AE CD ED AF CD DF
       ADBB AD 2A 00 12 CD 05 AE 23 3A 03 12  09E7
       ADC6 BC 38 08 20 F4 3A 02 12 BD 30 EE
       ADD1 11 1D AE CD 15 00 21 06 12 CD 20  1104
       ADDC AF 18 DA CD 06 00 11 29 AE CD 15
       ADE7 00 21 01 12 CD 4C AF 11 2E AE CD  18F8
       ADF2 15 00 21 03 12 CD 4C AF AF 32 04
       ADFD 12 32 05 12 32 06 12 C9 3A 04 12  1DAE
       AE08 86 32 04 12 3A 05 12 CE 00 32 05
       AE13 12 3A 06 12 CE 00 32 06 12 C9 20  2237
       AE1E 50 9D AD AA A4 A5 B3 B3 92 20 0D
       AE29 AB B7 B0 20 0D 20 9A A6 A4 20 0D  2C59
```

Prüfsumme über bisheriges EBS (ADB0...AFFF): FECD

L16.2.1a) Anfangsadresse: AFDF, Endadresse: ADEC. AFEC - AFDF = D. Es sind also 14 Bytes (!) zu transferieren.

```
RELPRU 4800  LD    IX,AFDF      DD 21 DF AF
       4804  LD    IY,1200      FD 21 00 12
       4808  LD    B,0E         06 0E
       480A  LD    A,(IX+00)    DD 7E 00
       480D  LD    (IY+00),A    FD 77 00
       4810  INC   IX           DD 23
       4812  INC   IY           FD 23
       4814  DJNZ  480A         10 F4
       4816  JP    1200         C3 00 12
```

Drei Fragen:
a) Weshalb dürfte das relokatierte Programm eigentlich nicht korrekt ablaufen? b) Weshalb tut es das doch? c) Was bewirkt der Sprung in 4816?
Zu a) Die Adresse des Textanfangs (AFE5) wurde nicht in 1206 geändert. Zu b) Das verschobene Programm arbeitet noch mit dem alten String. Zu c) Das verschobene Programm wird aufgerufen.

L16.2.1b) Es soll vorwärts (in Richtung des Programmablaufs) verschoben werden. Dazu müssen wir hinten (bei der höchsten Adresse) beginnen:

```
       4900  LD    IX,1204      DD 21 04 12
       4904  LD    B,05         06 05
       4906  LD    A,(IX+00)    DD 7E 00
       4909  LD    (IX+03),A    DD 77 03
       490C  DEC   IX           DD 2B
       490E  DJNZ  4906         10 F6
       4910  JP    DUMP12       C3 DF AF
```

L16.2.1c) Beim Rückwärtsverschieben muß man vorn beginnen:

```
4A00  LD    IY,1203       FD 21 03 12
4A04  LD    B,05          06 05
4A06  LD    A,(IY+00)     FD 7E 00
4A09  LD    (IY-03),A     FD 77 FD
4A0C  INC   IY            FD 23
4A0E  DJNZ  4A06          10 F6
4A10  JP    DUMP12        C3 DF AF
```

Wir hätten dasselbe natürlich auch mit LD IY,1200, LD A,(IY+03) und LD (IY+00),A erreicht, doch sollte wenigstens hier einmal mit negativem Offset gearbeitet werden.

```
L16.2.3 4B00  LD    HL,1203       21 03 12
        4B03  LD    DE,1200       11 00 12
        4B06  LD    BC,0005       01 05 00
        4B09  LDIR                ED B0
        4B0B  JP    DUMP12        C3 DF AF
```

L16.3.2 a) 10EF ... 1000 (rückwärts!)
b) 12EF ... 1200   c) 12EF

```
12E2  BC 9A   3. Probedaten
12E4  78 56   2. Probedaten
12E6  34 12   1. Probedaten
12E8  1B 30   4. Rücksprungadresse
12EA  13 30   3. Rücksprungadresse
12EC  0B 30   2. Rücksprungadresse
12EE  03 30   1. Rücksprungadresse
```

```
L16.4.2 2000  ..    ...           .. ..  wie P16.4.1
        2013  LD    D,H           54
        2014  LD    E,L           5D
        2015  EX    AF,AF'        08
        2016  ADC   HL,HL         ED 6A
        2018  ADC   HL,DE         ED 5A
        201A  EX    AF,AF'        08
        201B  CALL  PAUSWK        CD D4 AF
        201E  CALL  SBMONI        CD B3 AF
        2021  JR    LABEL         18 E5
```

```
L16.6.4 4000  LD    A,01          3E 01
        4002  LD    DE,6801       11 01 68
        4005  CALL  TIMST         CD 33 00
        4008  LD    A,(E005)      3A 05 E0
        400B  LD    H,00          26 00
        400D  LD    L,A           6F
        400E  LD    DE,0006       11 06 00
        4011  AND   A             A7
        4012  SBC   HL,DE         ED 52
        4014  JR    NC,4012       30 FC
        4016  CCF                 3F
```

```
        4017  ADC   HL,DE           ED 5A
        4019  LD    A,L             7D
        401A  INC   A               3C
        401B  CALL  OUTBYT          CD 30 AF
        401E  CALL  PAUSWK          CD D4 AF
        4021  CALL  LETNL           CD 06 00
        4024  JR    4008            18 E2
```

L16.7.2 a) L2 ist das Label mit dem Wert 412A b) 412A + 2 = 412C c) Eine Hexadezimalzahl, z.B. 00 d) Weil das B-Register ab hier eine innere Schleife kontrollieren soll. e) 412D ... 4145 f) 100d g) Alle sechs.

```
L16.8.3 2000  LD    B,A0            06 A0
        2002  LD    HL,1200         21 00 12
LOOP    2005  LD    A,R             ED 5F
        2007  LD    (HL),A          77
        2008  INC   HL              23
        2009  LD    A,33            3E 33
        200B  CALL  WARTEN          CD B8 AF
        200E  DJNZ  LOOP            10 F5
        2010  JP    DUMP12          C3 DF AF
```

(Ablaufdauer so etwa 40 Sekunden!)

```
L16.8.4 2000  LD    B,A0            06 A0
        2002  LD    HL,1200         21 00 12
        2005  LD    A,R             ED 5F       ;    9 Tz
        2007  LD    (HL),A          77          ;+   7 Tz
        2008  INC   HL              23          ;+   6 Tz
        2009  DJNZ  2005            10 FA       ;+  13 Tz
        200B  JP    DUMP12          C3 DF AF    ;=  35 Tz
```

Die 160 abgelesenen Zahlen zeigen, daß in R je Schleife um 5 weitergezählt wird. Da die Schleife - wie berechnet (siehe Z80-Handbuch) - 35 Taktzyklen lang ist, wird in R offenbar bei jedem 7. Zyklus um eins weitergezählt (inkrementiert).
Anregung: Bauen Sie in die Schleife an sich sinnlose aber unschädliche Befehle ein, deren Bearbeitungsdauer (in Taktzyklen) Sie dem Handbuch entnehmen. Kontrollieren Sie, ob sich auch dann die obige Rechnung bestätigt.

L16.9.2 Sie könnten dabei herausfinden:

```
Bit7: S-Flag, gesetzt bei "negativem" Ergebnis
Bit6: Z-Flag, gesetzt bei Ergebnis 0 (bzw. gleich)
Bit5: ?       (unbenutzt??)
Bit4: H-Flag, Halbbyte-Übertrag
Bit3: ?       (unbenutzt??)
Bit2: P/V-Flag, Parität und Überlauf
Bit1: N-Flag, vom Prozessor intern benutzt
Bit0: C-Flag, Übertragsbit, Carry.
```

```
L16.9.3 2000  LD   B,08        06 08     ;8 Binärst.
        2002  LD   D,00        16 00     ;Einsen zähl.
        2004  CALL INBYTE      CD 57 AF
        2007  RL   A           CB 17     ;Bit7 -> Cy
        2009  JR   NC,200C     30 01     ;wenn Cy = 0
        200B  INC  D           14        ;Eins zählen
        200C  DJNZ 2007        10 F9
        200E  CALL PRINTS      CD 0C 00
        2011  LD   A,D         7A
        2012  CALL OUTBYT      CD 30 AF
        2015  CALL LETNL       CD 06 00
        2018  JR   2000        18 E6
```

L17.1.2 a) 3004, 3013, 301C, 302F

b)

Elemente:	4	5	6	7	?
Ablaufzeit:	.2s	4s	30s	ca. 5 min	?

L18.3.2 Ändern Sie P18.3.1 ab Speicherstelle 400D so ab:

```
        400D  PUSH HL          E5
        400E  CALL TONAB       CD B2 02
        4011  POP  HL          E1
        4012  INC  HL          23
        4013  LD   A,H         7C
        4014  CP   12          FE 12
        4016  JR   NZ,400D     20 F5
        4018  CALL SBMONI      CD B3 AF
        401B  JR   4000        18 E3
```

Verändern Sie auch (4002) und (4015).

```
L19.1.2 2000  CALL CLS         CD 9F AF     ;1.
        2003  LD   B,10        06 10        ;2.
        2005  LD   IX,1300     DD 21 00 13  ;3.
        2009  LD   H,(IX+00)   DD 66 00     ;4.
        200C  LD   L,(IX+01)   DD 6E 01     ;5.
        200F  LD   A,(IX+02)   DD 7E 02     ;6.
        2012  LD   (HL),A      77           ;7.
        2013  INC  IX          DD 23        ;8.
        2015  INC  IX          DD 23
        2017  INC  IX          DD 23
        2019  DJNZ 2009        10 EE        ;9.
        201B  CALL PAUSWK      CD D4 AF     ;10.
        201E  RST  00          C7
```

```
L19.2.6 = TBPOS3   (Prüfsumme: D540)
15E0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0
15EE F0 F0 F0 F0 F0 F0 F0 FC F0 F0 F0 F0 F0 F0
15FC F0 F0 F0 F0 F0 F0 FE FF FD F0 F0 F0 F0 F0
160A F0 F0 F0 F0 F0 F0 FA FF F5 F0 F0 F0 F0 F0
1618 F0 F0 F0 F0 F0 F0 F8 FF F4 F0 F0 F0 F0 F0
1626 F0 F0 F0 F0 F0 F8 FF FF FF F4 F0 F0 F0 F0
1634 F0 F0 F0 F0 F8 FF FF FF FF FF F4 F0 F0 F0
```

```
1642 F0 F0 F0 F8 FF F7 FA FF F5 FB FF F4 F0 F0
1650 F0 F0 F0 F0 FB FD FA FF F5 FE F7 F0 F0 F0
165E F0 F0 F0 F0 F0 FB FF FF FF F7 F0 F0 F0 F0
166C F0 F0 F0 F0 F0 F0 FA FF FF F4 F0 F0 F0 F0
167A F0 F0 F0 F0 F0 F8 FF FF FF FF FC F0 F0 F0
1688 F0 F0 F0 F0 F8 FF F7 F0 F0 F3 FF FD F0 F0
1696 F0 F0 F0 F8 FF F3 F0 F0 F0 F0 FE FF F1 F0
16A4 F9 F4 FC FF F1 F0 F0 F0 F0 FE FF F1 F0 F0
16B2 F0 FB F7 F0 F0 F0 F0 F0 F0 F2 FD F4 F0 F0
Dazu gehört
POS3      2039  LD    IY,TBPOS3 FD 21 E0 15
          203D  CALL  BILD      CD 06 20
          2040  HALT            76          ;(RET)

L19.4.1   5000  LD    HL,1FFE   21 FE 1F
          5003  LD    DE,16DC   11 DC 16
          5006  LD    BC,00AA   01 AA 00
          5009  LDIR            ED B0
          500B  RST   00        C7
b) In 16E1                    JP   2041  ->  JP   171F
In 170B, 1713, 171B           CALL 2006  ->  CALL 16E4
In 1727, 173F, 1757, 176F CALL 2029  ->  CALL 1707
In 172D, 1739, 1745, 1751,
   175D, 1769, 1775, 1781 CALL 2031  ->  CALL 170F
In 1733, 174B, 1763, 177B CALL 2039  ->  CALL 1717
Prüfsumme (1400 ... 1785) = 2D6FD
Startadresse: 16DC

L22.1.2   PDUMP (2000 ... 20BA) -> (ACF5 ... ADAF)
In ACFB                       CALL 2046  ->  CALL AD3B
In AD0A, AD0E, AD17           CALL 2068  ->  CALL AD5D
In AD11, AD1A                 CALL 208A  ->  CALL AD7F
In AD2D, AD69, AD75           CALL 208C  ->  CALL AD81
In AD70                       JP   208C  ->  JP   AD81
In AD31, AD37                 CALL 207E  ->  CALL AD73
In ACF5                       11 92 20   ->  11 87 AD
In AD24                       FA 42 20   ->  FA 37 AD
In AD50                       11 AA 20   ->  AA 9F AD
In AD59                       32 20 20   ->  32 15 AD
Prüfsumme (ACF5 ... ADAF) = 5CD0

b)        VERSCH (3000 ... 3078) -> (AC7C ... ACF4)
In AC7C                       11 56 30   ->  11 D2 AC
In AC9A                       11 72 30   ->  11 EE AC
In ACB6                       FA 4B 30   ->  FA C7 AC
Prüfsumme (AC7C ... ACF4) = 3405
```

D Überblick über die EBS-Routinen und -Programme

(M) besagt, daß dieses Programm auch über MENUE erreichbar ist.

CLS Clear screen CALL AF9F → RET
Alle Register geschützt. Vgl. 14.5

DEZHEX (M) JP AE34 → Schleife
Verwandelt Dezimalzahlen < 065536 in Hexadezimalzahlen Vgl. 15.5

DISASS (M) JP B000 → Schleife
Disassembliert im Speicher stehende Maschinenprogramme. Ausgabe: Display oder Display & Printer. Vgl. 23..

DUMP12 JP AFDF → Monitor
Entspricht dem Monitor-Befehl D1200 Vgl. 8.1

HEXDEZ (M) JP AE92 → Schleife
Verwandelt vierstellige Hexadezimalzahlen in Dezimalzahlen. Vgl. 15.1

INBYTE CALL AF57 → RET
Liest ein Byte von der Tastatur ein. Gibt Echo auf dem Bildschirm. Eingabe steht im Akku. Vgl. 14.4

IN6HEX CALL AF47 → RET
IN4HEX CALL AF4C → RET
IN2HEX CALL AF51 → RET
Einlesen einer n-stelligen Hexadezimalzahl mit Echo und Ablage im Speicher. HL muß zu Anfang auf den Speicherplatz zeigen, in dem das erste Byte abgelegt werden soll. HL wird je Byte einmal dekrementiert. HL und A ungeschützt. Vgl. 14.5

KYECH CALL AFA7 → RET
Wartet mit blinkendem Cursor auf eine Eingabe. Nach Eingabe Echo und Rückkehr mit ASCII der gedrückten Taste im Akku. Vgl. 14.3

MENUE JP AA32 → Auswahl
Vgl. 22.4

MSGNL = MSG + LETNL CALL AFED → RET
AF ungeschützt Vgl. 7.3

NULL12 CALL AFF4 → RET
Setzt Speicher 1200 ... 129F auf null. A, B und HL ungeschützt. Vgl. 6.3

OFFSET (M) JP AB47 → Schleife
Berechnet und codiert Sprungweiten. Vgl. 22.3

OUTBIN CALL AEF8 → RET
Gibt den Akkuinhalt als Binärzahl aus. AF ungeschützt. Vgl. 14.8

OUTBYT CALL AF30 → RET
Gibt den Akkuinhalt als Hexadezimalzahl aus. AF ungeschützt.
Vgl. 14.6

OUT6HX CALL AF20 → RET
OUT4HX CALL AF25 → RET
OUT2HX CALL AF2A → RET
Gibt eine n-stellige Hexadezimalzahl auf dem Bildschirm aus. HL muß am Anfang auf den Speicherplatz zeigen, aus dem das erste Byte ausgegeben werden soll. Nach jedem Byte wird HL dekrementiert. HL und A ungeschützt. Vgl. 14.7

PAUSKY CALL AFD9 → RET
Diese Subroutine hält den Programmablauf an, bis eine Taste gedrückt wird. Wo sie in schnellen Programmen ignoriert wird, benutze man PAUSWK. AF ungeschützt. Vgl. 8.2

PAUSWK CALL AFD4 → RET
Wie PAUSKY, jedoch mit vorgelegter Verzögerung. Vgl. 10.5

PDUMP (M) JP ACF5 → Monitor
Druckt Speicherauszug. Vgl. 21.2

PRUESU (M) JP ADB0 → Schleife
Berechnet Prüfsummen. Vgl. 16.1

SBMONI CALL AFB3 → RET,
wenn SHIFT & BREAK gedrückt, → Monitor
AF ungeschützt. Vgl. 10.6

VERIFY (M) JP ABD7 → Schleife
Überprüft Speicherinhalt byteweise. Vgl. 22.2

VERSCH (M) JP AC7C → Schleife
Verschiebt Speicherblöcke. Vgl. 22.1

WARTEN CALL AFB8 → RET
Warteschleife, deren Ablaufdauer vom Akkuinhalt bestimmt wird.
Vgl. 10.5

Stichwortverzeichnis

Z

VOGEL-BUCHVERLAG WÜRZBURG



James, Mike

Der Weg zur ZX Spectrum-Meisterschaft

Reihe HC –
Mein Home-Computer

216 Seiten,
19 Abbildungen,
30,– DM, 1985
ISBN 3-8023-**0810**-7

Durch das Erscheinen der Mikrodrives und der Interfaces I und II wurder der ZX Spectrum noch vielseitiger einsetzbar. Wie man BASIC-Programme durch Maschinencode-Routinen erweitert, die technischen Möglichkeiten des ZX Spectrum ganz ausnutzt und aktuelle Peripherie-Einheiten erfolgreich einsetzt – das erfahren Sie hier sehr ausführlich. Kenntnisse in BASIC werden vorausgesetzt.

Jones, Robin
Fairhurst, Michael

Start in die Künstliche Intelligenz für Ihren ZX-Spectrum

Reihe HC –
Mein Home-Computer

ca. 192 Seiten,
ca. 30,– DM,
ISBN 3-8023-**0862**-X

erscheint voraussichtlich im Mai 1985

Das Buch gibt eine Einführung in einige Techniken der künstlichen Intelligenz (KI). Es beschränkt sich auf solche Aspekte von KI, die leicht auf dem ZX-Spectrum anzuwenden sind. Voraussetzung zum Gebrauch des Buchs sind Kenntnisse in BASIC; eine erste Bekanntschaft mit dem Maschinencode kann ebenfalls nützlich sein, denn viele der Routinen kann man dann in Maschinencode umsetzen.

Anhand aktionsgeladener und teils kniffliger Spiele werden Sie mit Programmiertechniken und Tricks vertraut gemacht. Wenn Maschinencodeprogramme eingesetzt sind, werden hinreichende Erläuterungen gegeben. Das intensive Arbeiten mit diesem Buch wird Einsteigern wie Fortgeschrittenen gleichermaßen Freude bereiten, neue Perspektiven eröffnen und zum kreativen Computern anregen.



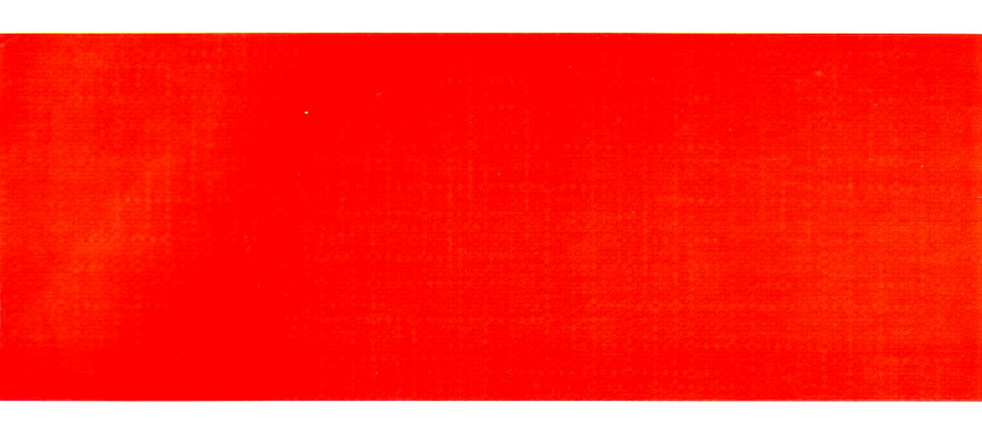
Guss, Thomas

Was der ZX Spectrum alles kann

Reihe HC –
Mein Home-Computer

160 Seiten,
zahlr. Abbildungen
und Listings,
28,– DM, 1984
ISBN 3-8023-**0762**-3

Schneller erfolgreich durch Computer-Bücher



Die Sprache des Mikroprozessors Z80 (bzw. Z80A) ist überall dieselbe – unabhängig davon, in welchem Computer der Z80 arbeitet. Der Verfasser wählte als Beispielmodell den MZ-700, weil er ein gängiger Mikrocomputer ist, sich ausgezeichnet zum Arbeiten in der Maschinensprache eignet und hervorragend dokumentiert ist. Auch wer mit einem anderen Z80-Computer arbeitet, wird viele brauchbare Beispiele und Anregungen finden. In jedem Fall ist es nützlich, wenn der Leser bereits Kenntnisse im Programmieren hat.

Dieses Buch vermittelt die wichtigsten Grundbegriffe und Z80-Befehle, unterstützt beim Zurechtfinden in den Handbüchern und Kennenlernen gängiger Programmstrukturen, gibt Anregungen für eigenes Arbeiten und zum Gebrauch von Dienstprogrammen, verrät viele nützliche Programmiertricks.



ISBN 3-8023-0830-1